AÎURI: UM PORTAL PARA MINERAÇÃO DE TEXTOS INTEGRADO A GRIDS COMPUTACIONAIS

Antonio Anddre Serpa da Silva

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA CIVIL

Aprovada por:

_____
Prof. Nelson Francisco Favilla Ebecken, D.Sc

_____
Dra. Myrian Christina de Aragão Costa, D.Sc

_____
Prof. Beatriz de Souza Leite Pires de Lima, D.Sc

_____
Prof. Mario Antonio Ribeiro Dantas, Ph.D

RIO DE JANEIRO, RJ - BRASIL
DEZEMBRO DE 2007

DA SILVA, ANTONIO ANDDRE SERPA

Aîuri: Um Portal para Mineração de Textos Integrado a Grids Computacionais [Rio de Janeiro] 2007

X, 155p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia Civil, 2007)

Dissertação - Universidade Federal do Rio de Janeiro, COPPE

1. Mineração de Textos

2. Grids Computacionais

3. Portal

I. COPPE/UFRJ II. Título (série)

Agradecimentos

À minha orientadora, Professora Myrian Costa, pelo inestimável apoio, entusiasmo, amizade e confiança, que foram determinantes para o sucesso deste trabalho.

Ao meu orientador, Professor Nelson Ebecken, pela sua confiança e apoio ao trabalho desenvolvido.

À minha amiga Carla Lage, pela sua amizade e por seu decisivo incentivo para que eu iniciasse este curso de mestrado.

Ao meu amigo Leonardo Ferreira, pelo amigo que é e pelas várias horas na revisão do texto deste trabalho, com sugestões que foram decisivas para o sucesso do mesmo. Obrigado irmão!

À minha amiga Valeriana Roncero, por seu espírito de companheirismo e ajuda em momentos importantes.

Ao amigo Marco Aurélio Ribeiro Dantas, pela confiança depositada em meu trabalho, ao gerar os resultados de sua dissertação utilizando o sistema Aîuri.

Aos meus eternos amigos, Nilson Gonçalves, Rodrigo Medeiros, Sérgio Pinto e Alexander Meneguetti, por esta amizade que transcende as mais imponentes barreiras.

Ao Doutor Luiz Paulo Franca, por ser o amigo que muito me ensinou e muito me ajudou em momentos decisivos da minha vida profissional.

À banca examinadora pelas suas sugestões que contribuíram para o aprimoramento deste trabalho.

Ao NACAD pelo apoio e estrutura disponibilizada.

Aos amigos do NACAD, que sempre foram uma fonte de apoio e incentivo.

Ao EELA / CIEMAT, pela minha viagem à Venezuela, que possibilitou novas descobertas e o enriquecimento deste trabalho.

À Marinha do Brasil, instituição que sempre confiou em mim, pela oportunidade que

meu deu de realizar uma qualificação de tão alto nível.

Aos meus amigos da Marinha, Capitão-Mar-e-Guerra Luiz Augusto e Capitão-Tenente Mônica Fonte, por esses anos de tão sincera amizade. Ao Comandante Luiz Augusto, pelo seu apoio e compreensão quando era vice-diretor da PAPEM, tendo um papel decisivo para a minha entrada neste curso de mestrado.

Aos meus tios, Márcia Serpa, Margarete Serpa, Myrtis Serpa, Gutemberg Serpa, Lindemberg Serpa e Carlos Guerra por seu amor e apoio incondicionais nos momentos mais importantes da minha vida.

À minha prima Eliude Ferreira, por sua confiança, amizade, e sempre sábias e determinantes palavras. Muito obrigado, minha prima!

Aos meus queridos avós, Aderval Gomes Serpa, Josefa Ferreira Serpa e Antonio José da Silva, os quais sempre foram uma fonte de exemplos, e onde quer que estejam, me inspiram e intercedem por mim.

À minha mãe, Marta Serpa, por ser a pessoa mais importante da minha vida, com sua luz, firmeza, postura, dedicação e amor. De grandes batalhas participamos e juntos vencemos todas. Finalmente, é chegada a hora de colhermos os frutos. Eu te amo, minha mãe!

A Deus e a todos aqueles que me protegem e me guiam pelo caminho do bem.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

AÎURI: UM PORTAL PARA MINERAÇÃO DE TEXTOS INTEGRADO A GRIDS COMPUTACIONAIS

Antonio Anddre Serpa da Silva

Dezembro/2007

Orientadores: Nelson Francisco Favilla Ebecken
Myrian Christina de Aragão Costa

Programa: Engenharia Civil

O sistema Aîuri é um ambiente acadêmico cooperativo de alto desempenho que tem utilidade no ensino e pesquisas nas áreas de inteligência computacional, análise, avaliação e visualização de dados, integrado a ambientes de *grids* computacionais.

Um portal *Web* integrado a dois ambientes de *grid* computacional, foi desenvolvido para a utilização de algoritmos de mineração de textos. Os ambientes de *grid* utilizados são o Intragrid NACAD, administrado pelo Núcleo de Computação de Alto Desempenho (NACAD) da COPPE, que agrupa máquinas heterogêneas do laboratório em um *grid* com finalidade didática, e o *E-Infraestructure Shared Between Europe and Latin America* (EELA), que é uma infra-estrutura para o desenvolvimento e implantação de *grids* para uso científico, conectando a Europa e a América Latina.

O portal Aîuri é composto por 3 módulos principais. O primeiro módulo realiza as atividades de autenticação e carregamento de arquivos dos usuários. O segundo módulo é o responsável pelas tarefas de pré-processamento de dados textuais. No terceiro módulo são implementados os já consagrados algoritmos para mineração de textos, os classificadores *bayesiano* e de *ranqueamento linear*. Neste módulo, são também disponibilizadas as respectivas métricas dos algoritmos para que possam servir de objeto de análise pelo pesquisador.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

AÎURI: A WEB PORTAL FOR TEXT MINING INTEGRATED TO COMPUTATIONAL GRIDS

Antonio Anddre Serpa da Silva

December/2007

Advisors: Nelson Francisco Favilla Ebecken
Myrian Christina de Aragão Costa

Department: Civil Engineering

The Aîuri web portal is a cooperative academic environment, that will be of great use in research as well as teaching in the fields of computational intelligence, analysis, evaluation and visualization of non-structured data, integrated to computational grid environments.

A web portal, integrated to two distinct computational grid environments, was developed to be used as an interface for running text mining algorithms in a grid environment. The grid environments used are the Intragrid NACAD, deployed at the High Performance Computing Center (NACAD) at COPPE, that connects heterogeneous machines in a grid environment with academic purposes, and the E-Infrastructure Shared Between Europe and Latin America (EELA), which is an infrastructure for the development and deployment of grids for scientific use, connecting Europe and Latin America.

The Aîuri portal is composed of three modules. The first one is responsible for user authentication and loading the user files. The second module is responsible for pre-processing of textual data. Finally, in the third module, two well-known text mining algorithms – bayesian and linear score classifiers – are implemented. Additionally, in this module the metric of both algorithms are available to be object of analysis by the researcher.

Sumário

1 Introdução **1**

2 Mineração de Textos **8**
2.1 Tipos de Abordagens de Dados 10
2.1.1 Análise Semântica 11
2.1.2 Análise Estatística 11
2.2 Etapas do Processo de Mineração de Textos 12
2.2.1 Coleta de Documentos 12
2.2.2 Pré-processamento 13
2.2.2.1 Análise Léxica 13
2.2.2.2 *Stopwords* 14
2.2.2.3 *Stems* 15
2.2.3 Preparação dos Dados 16
2.2.3.1 Cálculo de Relevância 17
2.2.3.2 Representação de Documentos 19
2.2.3.3 Seleção de Atributos 22
2.2.4 Extração de Padrões 24
2.2.5 Avaliação e Interpretação de Resultados 27
2.2.6 O Processo de Categorização 28
2.2.6.1 Algoritmo *Naive-Bayes* 30
2.2.6.2 Algoritmo de Ranqueamento Linear 31
2.3 Ferramentas de Mineração de Textos 34
2.3.1 Ferramentas de Domínio Público 34
2.3.1.1 *Text Mine* 34
2.3.1.2 *TMSK - Text-Miner Software Kit* 35
2.3.1.3 *RIKTEXT - Rule Induction Kit for Text* 35
2.3.1.4 *Intext Software* 36
2.3.2 Ferramentas Comerciais 36
2.3.2.1 *TextSmart* 36
2.3.2.2 *STATISTICA Text Miner* 37
2.3.2.3 *SQL Server 2005* 38
2.3.2.4 *LexiQuest Categorize* 39

3 *Grids* Computacionais e Portais **40**
3.1 Benefícios 44
3.2 Desafios 45

3.3 Componentes . . . 45
3.3.1 Recursos . . . 45
3.3.2 Arquitetura . . . 46
3.3.2.1 Camada de Fábrica . . . 46
3.3.2.2 Camada de Conectividade . . . 47
3.3.2.3 Camada de Recursos . . . 48
3.3.2.4 Camada de Serviços Coletivos . . . 49
3.3.2.5 Camada de Aplicações . . . 50
3.4 Serviços Web . . . 50
3.4.1 Acesso a Serviços . . . 51
3.5 Serviços Grid . . . 52
3.5.1 Descoberta de Serviços . . . 53
3.5.2 Autenticação e Autorização . . . 54
3.5.3 Privacidade de Dados . . . 54
3.5.4 Composição de Serviço . . . 55
3.6 *Open Grid Forum* (OGF) . . . 55
3.6.1 OGSI . . . 56
3.6.2 OGSA . . . 57
3.6.2.1 WSRF . . . 58
3.7 Globus Toolkit . . . 60
3.7.1 *Globus Toolkit 4* . . . 61
3.8 *Grid EELA* . . . 62
3.9 Portais de *Grid* . . . 64

4 O ambiente Computacional **68**
4.1 Eclipse . . . 68
4.2 Java . . . 69
4.3 Java Server Pages (JSP) . . . 70
4.4 XML . . . 71
4.5 Padrões de Projeto . . . 72
4.6 Arquitetura MVC . . . 73
4.7 Arquitetura Modelo 2 . . . 75
4.8 *Frameworks* . . . 76
4.9 *Struts Framework* . . . 77
4.10 Linguagem de Modelagem Unificada . . . 79
4.11 O Processo Unificado de Desenvolvimento de Software . . . 80
4.11.1 Características . . . 81
4.11.1.1 Processo Orientado por Casos de Uso . . . 81
4.11.1.2 Processo Centrado na Arquitetura . . . 82
4.11.1.3 Processo Iterativo e Incremental . . . 83
4.12 Modelagem do Sistema . . . 84
4.12.1 Diagrama de Casos de Uso . . . 85
4.12.1.1 Descrição dos Casos de Uso . . . 85
4.12.2 Diagrama de Pacotes . . . 87
4.12.3 Diagrama de Classes . . . 87
4.13 Funcionamento do Sistema . . . 93
4.14 Atividades de Integração dos Ambientes de *Grid* . . . 103

4.14.1 Integração no *Grid* GT4 103
4.14.1.1 O ambiente do GT4 104
4.14.1.2 Criação das Classes Serviço *Web* 104
4.14.1.3 Publicação do Serviço *Web* 105
4.14.1.4 Publicação do Serviço *Grid* 107
4.14.2 Integração no *Grid* EELA 108
4.14.2.1 Utilização do AMGA 110
4.14.2.2 Utilização do GSAF 113
4.14.2.3 Submissão de jobs 115

5 Testes do Portal Aîuri **119**
5.1 Textos Utilizados 120
5.2 O Problema 120
5.3 Procedimentos para a Solução do Problema 120
5.3.1 Teste sem balanceamento 121
5.3.2 Teste com balanceamento 122
5.3.3 Padronizações do Textos para Uso no Ambiente 123
5.3.4 Avaliação dos Categorizadores 124
5.3.4.1 Categorizador *Bayesiano* 125
5.3.4.2 Categorizador Linear 132

6 Conclusões e Trabalhos Futuros **142**
6.1 Trabalhos Futuros 144

Apêndices **145**

A Inclusão de Novos Algoritmos **145**

Referências **148**

Abreviaturas

ASCII	American Standard Code for Information Interchange
CORBA	Common Object Request Architecture Broker
EELA	E-Infraestructure Shared Between Europe and Latin America
EI	Extração de Informações
EJB	Enterprise Java Beans
HTML	HyperText Markup Language
idf	Inverse Document Frequency
IE	Information Extraction
JDL	Job Description Language
JVM	Java Virtual Machine
KDD	Knowledge Discovery in Databases
KDT	Knowledge Discovery in Textual Databases
ML	Machine Learning
MPI	Message Passing Interface
MVC	Model View Controller
NACAD	Núcleo de Atendimento a Computação de Alto Desempenho da COPPE/UFRJ
NB	Naive Bayes
NLP	Natural Language Processing
ODBC	Open Data Base Connectivity
OGSA	Open Grid Service Architecture
OGSI	Open Grid Services Infrastructure
PDF	Portable Document Format
PU	Processo Unificado
RI	Recuperação da Informação
RMI	Remote Method Invocation
SQL	Structured Query Language
SSIS	SQL Server Integration Services
TCP/IP	Transmission Control Protocol/Internet Protocol
TDM	Text Data Mining
tf	Term Frequency
tf–idf	Term Frequency–Inverse Document Frequency
UML	Unified Modeling Language
VO	Virtual Organization
VSM	Vector Space Model
WSRF	Web Service Resource Framework
XML	eXtensible Markup Language

Capítulo 1

Introdução

O advento das técnicas de Mineração de Textos (*Text Mining*) [1–3] tornou possível a exploração de diversos tipos de textos em formato eletrônico que compõem o dia a dia de empresas e pessoas. Diariamente, inúmeras páginas e documentos contendo textos são disponibilizados na *Web*.

Em virtude do crescimento contínuo do volume de dados eletrônicos disponíveis, técnicas de extração de conhecimento automáticas tornam-se cada vez mais necessárias para manipular essa gigantesca massa de dados. O principal objetivo das técnicas de mineração de textos é a manipulação de documentos em formato texto que se encontram de forma não-estruturada. De fato, as aplicações deste gênero fornecem uma nova dimensão das informações e que, se bem exploradas, podem se tornar um diferencial no domínio do negócio onde for aplicada. Desta maneira, um processo manual de avaliação e classificação de documentos torna-se inviável, na medida em que gera resultados baseados em uma visão limitada dos conteúdos dos documentos, ou demanda muito tempo para ser realizado.

Em um outro contexto, é fato que as instituições e empresas vêm, com o passar dos anos, acumulando uma quantidade significativa de dados e informações. Com o advento da Internet, adquirir informação se tornou algo trivial. Porém, como tirar o melhor proveito possível da informação acumulada e processá-la de forma a produzir resultados tangíveis?

Mineração de Textos, também chamada de *mineração de dados textuais* ou *descoberta de conhecimento de bases de dados textuais* é um campo novo e multidisciplinar que inclui conhecimentos de áreas como Informática, Estatística, Linguística e Ciência Cognitiva. Mineração de textos tem por objetivo extrair regularidades, padrões ou tendências de grandes

volumes de textos em linguagem natural, normalmente para objetivos específicos. Inspirado pelo *data mining* ou mineração de dados, que destina-se a descobrir padrões emergentes de banco de dados estruturados, a mineração de textos é destinada à extração de conhecimentos úteis de dados não estruturados ou semi-estruturados.

Na literatura pode-se encontrar diferentes definições dos termos "Descoberta do Conhecimento" ou "Descoberta do Conhecimento em Bases de Dados" (também conhecido como KDD - *Knowledge Discovery in Databases*), como também para mineração de dados. A análise de dados em KDD, busca essencialmente, a descoberta de padrões e conexões com dados até então desconhecidos. Por dados, pode-se entender uma quantidade de fatos, que podem estar em um banco de dados, como também podem ser dados contidos em um simples documento textual. Algumas das características que podem ser usadas para medir a qualidade dos padrões encontrados nos dados são:

(i) Ser intelegível: o padrão encontrado deve ser de fácil entendimento;

(ii) Ser válida no contexto das métricas estatísticas;

(iii) Ser inovador: deve expressar alguma novidade;

(iv) Ser útil: o padrão encontrado deve ter alguma utilidade no contexto do domínio do problema.

Desta maneira, a expressão "potencialmente utilizáveis" significa que os padrões encontrados por uma determinada aplicação, seja ela de mineração de dados ou mineração de textos, deve, de alguma forma, gerar benefícios tangíveis para o usuário.

Segundo Hotho et al. [2], a descoberta do conhecimento é um processo que está especificado em algumas etapas de processamento que devem ser aplicadas ao conjunto de dados a ser utilizado pelo usuário, de maneira que se possa obter algum conhecimento ainda não verificado. Essas etapas devem ser realizadas de forma iterativa e muitas delas, geralmente, precisam de uma intervenção direta do usuário.

Além da necessidade de entender o problema e possuir um entendimento geral das tarefas envolvidas, atenção especial deve ser dada à etapa de preparação de dados. Esta etapa é muito importante, principalmente quando as técnicas de mineração de textos são aplicadas, visto que os dados a serem manipulados por suas aplicações devem ser convertidos para um

formato compatível com os algoritmos. Assim, da mesma maneira que os algoritmos de mineração de dados são usados para construir e testar o modelo gerado, para que depois os dados sejam disponibilizados para análise, o mesmo ocorre com as aplicações de mineração de textos.

O caráter interdisciplinar das atividades de mineração de dados e de mineração de textos, conduz as pesquisas para três importantes áreas: bancos de dados, aprendizado de máquina e estatística.

Bancos de dados são necessários na medida em que são capazes de armazenar grandes quantidades de dados de maneira muito eficiente. Neste contexto, um banco de dados não representa somente uma mídia para um armazenamento consistente e de fácil acesso, mas é algo de grande valia para os pesquisadores, uma vez que pode ser utilizado por algoritmos que acessem esta base. Uma visão geral sobre mineração de dados em banco de dados pode ser encontrada em [4].

Aprendizado de Máquina, ou *Machine Learning* (ML), é uma área da inteligência artificial concebida para desenvolver técnicas que tornem o computador capaz de "aprender" a partir de um conjunto de dados. Alguns métodos de aprendizado de máquina podem ser discutidos em [5].

A Estatística tem sua fundamentação em teorias matemáticas e lida, basicamente, com a análise de dados empíricos. Com base na teoria de probabilidade, aleatoriedades e incertezas são modeladas. Muitos métodos estatísticos são usados na área de KDD. Mais detalhes nesta área podem ser obtidos em [6–8].

Dentre as várias perspectivas desta área, o presente trabalho aborda o problema de classificação/categorização automática de textos.

A classificação de documentos em formato texto é uma tarefa realizada por indivíduos, que possuem conhecimento no domínio de interesse, que lêem e classificam os documentos em categorias temáticas pré-definidas. É notório o fato de que humanos possuem uma capacidade limitada para lidar com uma elevada quantidade de documentos, fato este que levou ao desenvolvimento das técnicas de mineração de textos. Estas atividades consistem em classificar automaticamente textos em linguagem natural, em categorias temáticas pré-definidas, como foi demonstrado em [9]. Níveis satisfatórios de desempenho já foram alcançados em relação a classificações realizadas por profissionais devidamente preparados [10]. Alguns exemplos da aplicação de técnicas de mineração de textos são:

(i) Filtragem de emails (*spam*) [11];

(ii) Filtragem de conteúdo impróprio da Internet – violência, pornografia e racismo – em ambientes educacionais [12];

(iii) Classificação de notícias jornalísticas por assunto [13–15].

Outros exemplos de aplicações podem ser encontrados em [10] e [16].

A abordagem dominante, até o final da década de 1980, envolvia a construção de classificadores especialistas, ou seja, construía-se manualmente um conjunto de regras lógicas a partir do conhecimento do especialista do domínio, de maneira a classificar documentos nas categorias sob consideração (Sebastiani [10]).

Assim, o especialista, para cada categoria temática no domínio do problema, deve prever todas as combinações de palavras cuja co-ocorrência leve a um documento a ser classificado na mesma categoria.

Estes tipos de classificadores, mesmo apresentando altos níveis de desempenho, exigem elevado poder computacional, e são completamente dependentes do domínio da aplicação.

Porém, na década de 1990, com o aumento da produção e disponibilidade de documentos em meio eletrônico, uma nova abordagem surgiu, baseada no aprendizado de máquina, que suplantou a abordagem anterior.

Segundo Sebastiani [10], esse novo paradigma usa como base um conjunto de documentos pré-classificados em diversas categorias temáticas do domínio de interesse. A partir daí, um processo de indução é aplicado para identificar as características que diferenciam as diversas categorias entre si.

Com base neste processo de indução, um modelo de classificação é construído, sendo usado para classificar documentos ainda não vistos. Para isso, são realizadas comparações entre as características presentes no novo documento com os padrões já mapeados dos documentos já conhecidos, e que participaram da construção do modelo. Denomina-se este processo de aprendizagem supervisionada, uma vez que o processo de aprendizagem – construção do modelo de classificação – é supervisionado por exemplos de documentos cujas categorias são conhecidas.

O presente trabalho foi desenvolvido com base nos conceitos utilizados em aprendizado de máquina. A técnica de categorização automática de textos consiste fundamentalmente em:

(i) Disponibilizar um conjunto de documentos pré-classificados nas diversas categorias de interesse;

(ii) Transformar a informação textual contida nos documentos para um formato que possa ser manipulado computacionalmente pelos algoritmos de classificação;

(iii) Escolher os termos mais relevantes do conjunto de documentos, isto é, os que permitem melhor discriminação entre as categorias temáticas sob estudo;

(iv) Definir os pesos para os termos dos documentos, de maneira a possibilitar uma maior discriminação entre as categorias consideradas;

(v) Estimar o modelo de classificação;

(vi) Avaliar a efetividade do modelo estimado.

Todas estas etapas são detalhadas no capítulo 2. Para mais detalhes sobre a técnica de categorização de textos, ver [10].

Outro fator a ser considerado, é que o *hardware* teve uma redução considerável em seu custo, de modo que atualmente é possível a aquisição de um computador, com uma considerável capacidade de processamento, por algumas centenas de dólares. Assim, a impressionante melhoria de desempenho que as redes de computadores vêm experimentando levou à idéia de se utilizar computadores independentes conectados entre si como plataforma para execução de aplicações paralelas, originando a área de Computação em *Grid*.

Grids Computacionais nasceram da comunidade de Processamento de Alto Desempenho, motivada pela idéia de se utilizar computadores independentes e geograficamente dispersos como plataforma de execução de aplicações paralelas [17]. Porém, com a evolução da pesquisa sobre *Grids* Computacionais e tecnologias utilizadas pela indústria para computação distribuída, houve, naturalmente, uma convergência entre o mundo acadêmico e empresarial. Assim, a idéia é prover uma infra-estrutura que viabilize serviços sob demanda, permitindo uma maior colaboração entre várias instituições, através do compartilhamento de serviços e recursos e, utilizando mecanismos que facilitem a interoperabilidade.

Os atrativos desta idéia incluem a possibilidade de fornecimento de qualquer serviço computacional sob demanda, o qual pode ser composto dinamicamente por outros serviços e agregar recursos localizados em várias instituições distintas e geograficamente dispersas.

Além disso, os recursos podem ser alocados em uma quantidade enorme (e.g. milhares de computadores conectados via Internet) e por um custo muito menor do que alternativas tradicionais (baseadas em supercomputadores paralelos).

Uma outra idéia é que utilizando a tecnologia de *grids*, uma organização pode utilizar toda a capacidade de seu parque de máquinas, unindo desde aplicações, servidores, bases de dados, capacidade de processamento, de maneira a disponibilizar tudo isso como um serviço virtualizado. Assim, usam-se os recursos computacionais de modo confiável e transparente, não importando em qual computador foi processado ou está armazenada a informação necessária.

Todavia, as vantagens que os *grids* oferecem não podem ser utilizadas em sua plenitude se o acesso a eles for complexo. O usuário necessita de um modo amigável e produtivo de interagir com este ambiente. Para isso foram desenvolvidos os Portais de *Grids*. Esses portais unem as características e facilidades que a Internet trouxe às empresas com a tecnologia dos *grids*. Com o uso de uma interface amigável, os usuários podem submeter novos procedimentos ao *grid*, como também acompanhar o processamento dos que já estão em execução. Um administrador pode vizualizar rapidamente o estado do *grid*, observando seu número de usuários, capacidade utilizada etc.

O foco deste projeto é a criação de um ambiente cooperativo acadêmico de alto desempenho que tem utilidade no ensino e pesquisa nas áreas de inteligência computacional, análise, avaliação e visualização de dados não-estruturados, com serviços de *grid* para mineração de textos, com a implementação de um portal *web* que receba requisições para serem executadas neste tipo de ambiente.

O sistema, doravante denominado Aîuri, é um portal *web* desenvolvido na linguagem de programação Java, que contém toda uma infra-estrutura para capacitá-lo a executar algoritmos em ambientes de *grids* computacionais. Estes algoritmos são capazes de determinar a qual classe um novo texto pertence baseado no conhecimento obtido de textos anteriores. Dentre as funcionalidades implementadas, pode-se citar:

(i) *Carregamento* de arquivos e certificados de usuários;

(ii) Geração de conjuntos de textos para treinamento e testes dos algoritmos;

(iii) Geração de métricas dos modelos gerados;

(iv) Geração dos modelos bayesiano e de ranqueamento linear;

(v) Visualização do status dos trabalhos submetidos.

Os *softwares* a serem integrados neste ambiente devem ser abertos e disponibilizados para a comunidade, que poderá ter acesso aos serviços residentes, assim como incluir novos serviços. A não utilização de *softwares* comerciais é justificada pelo alto custo apresentado.

A utilização de uma infra-estrutura de *grid* computacional visa agregar funcionalidades com características colaborativas, sendo permitida a utilização e/ou incorporação de novas estratégias, possibilitando, com isso, a pesquisa de novos algoritmos e diferentes abordagens para a solução de problemas de engenharia e ciências avançadas. Dentro deste contexto, o *software* desenvolvido foi projetado de modo a manter-se escalável, utilizando técnicas ao nível de padrões de projeto [18].

A abordagem sistêmica utilizada para a modelagem do *software* segue as técnicas preconizadas pelo Processo Unificado (PU) [18], de maneira a implementar as melhores práticas atuais de desenvolvimento de *software*.

A arquitetura é implementada utilizando a linguagem Java, o servidor de aplicações Tomcat v5.5, o *middleware* para *grids* computacionais Globus Toolkit 4, assim como o *framework* para desenvolvimento baseado no modelo MVC *Struts* [19], visando principalmente as manutenções futuras.

Organização da Dissertação

Este trabalho é estruturado como se segue. No capítulo 2 são apresentadas as técnicas, conceitos e algoritmos utilizados nas atividades de mineração de textos. Em seguida, no capítulo 3, ambientes de *grids* computacionais e portais e suas aplicações são apresentadas. No capítulo 4 são apresentados os ambientes computacionais nos quais o portal foi desenvolvido e as ferramentas utilizadas. O capítulo 5 é dedicado à apresentação de resultados dos testes do ambiente desenvolvido. No capítulo 6 são apresentadas as conclusões e perspectivas de trabalhos futuros. Finalmente, no apêndice A são detalhados os passos necessários para a inclusão de novos algoritmos na aplicação.

Capítulo 2

Mineração de Textos

Mineração de Textos, também conhecido como *Text Data Mining* (TDM) [20] e *Knowledge Discovery in Textual Databases* (KDT) [1] pode ser descrito como um processo de identificação de informações desconhecidas de uma coleção de textos [20]. Por informações desconhecidas, pode-se pensar em associações, hipóteses ou tendências que não estão explícitas no texto objeto de análise. Mooney e Nahm [21] descrevem mineração de textos como "uma procura por padrões em textos não-estruturados". Já Dorre et al. [22] afirmam que "mineração de textos aplica as mesmas funções de análise de mineração de dados para o domínio de informações textuais, baseado em sofisticadas técnicas de análise de textos que obtém informações de documentos não-estruturados", enquanto Tan [23] descreve mineração de textos como "o processo de extração de padrões interessantes e não triviais ou conhecimento de documentos textuais". Nenhum dos autores, todavia, enfatiza explicitamente a importância das novidades descobertas nas informações mineradas.

Hearst [20] faz uma das primeiras tentativas de estabelecer claramente o que de fato constitui mineração de textos, como também tenta distinguí-lo de recuperação da informação e mineração de dados. Neste artigo, a autora, metaforicamente, descreve mineração de textos como um processo de garimpagem de "pedras preciosas em uma rocha", chamadas *nuggets* (partes do conhecimento), a partir de uma grande quantidade de informações. Diz que mineração de textos é o processo de descoberta de informações até agora desconhecidas de um texto. Por exemplo, suponha que um documento estabelece uma relação entre os tópicos A e B, e outro documento estabelece uma relação entre os tópicos B e C. Esses dois conjuntos de documentos possivelmente estabelecem um novo relacionamento entre A e C

(possivelmente, porque não existe uma relação explícita entre *A* e *C*). A autora também faz considerações sobre tarefas como categorização de textos, clusterização etc, dizendo que estas não podem ser classificadas como mineração de textos, visto que não produzem nada de novo. Usando outra metáfora, ela descreve recuperação da informação como um processo de busca em documentos que contém alguma informação de interesse para o usuário, como "procurar uma agulha num palheiro". Uma vez que a informação procurada já esteja presente no texto, ela considera que nenhum novo "*nugget*" é revelado.

Mineração de textos é também muitas vezes confundido com mineração de dados, com algumas pessoas descrevendo mineração de textos como uma simples extensão de mineração de dados, aplicada a dados não-estruturados. Hearst [20] discute isto dizendo que mineração de dados não é "garimpagem" somente, porém uma descoberta semi-automatizada de padrões ou tendências em grandes bancos de dados, que auxiliam na tomada de decisões e que nenhum fato novo é estabelecido durante este processo de descoberta.

Kroeze et al. [24] discutem algumas definições de Hearst [20], argumentando que a novidade definida como "pedras preciosas em uma rocha", os *nuggets*, é uma contradição nos seus próprios termos, porque essas pedras já estão presentes na rocha e, portanto, nada de novo está sendo extraído. Embora a metáfora de Hearst [20] possa não ser completamente apropriada, deve-se reconhecer que a sua ênfase-chave está na novidade. Kroeze et al. [24] também estendem-se sobre os argumentos de Hearst [20], introduzindo um novo conceito chamado "semi-novidade". Eles classificam dados/recuperação da informação como não-novidades, descoberta de conhecimento (mineração de dados padrão, metadata mineração, e mineração de texto padrão) como sendo semi-novo, e introduzem um novo tipo de mineração inteligente de texto, chamada de processo de investigação. Eles definem *intelligent text mining* como um processo de criação de conhecimento automático. Realçam, também, que as técnicas de inteligência artificial, que ajudam a simular a inteligência humana, são críticas para realizar a mineração inteligente de textos.

Alguns também tendem a confundir mineração de texto com Extração de Informações (IE). A IE trata da extração de fatos sobre entidades pré-especificadas, eventos ou relações de fontes de texto irrestritas. Pode-se pensar na extração de informações como a criação de uma representação estruturada de uma informação presente no texto [25]. Contudo, a extração de informações está diretamente envolvida no processo de mineração de textos. Por exemplo, uma abordagem utilizada na mineração de textos semi-estruturados na *web*, é usar

técnicas de extração para converter documentos em uma coleção de dados estruturados e depois aplicar técnicas de mineração de dados para análise.

A mineração de textos representa, de fato, um avanço em relação à recuperação de textos. É uma área de pesquisa relativamente nova que está mudando a ênfase das tecnologias de informações baseadas em texto, saindo de um nível de recuperação e extração, para um nível mais alto, como análise e exploração dos dados. Considerando a grande quantidade de informações disponíveis em forma de texto atualmente, ferramentas que realizem atividades nesta área de forma automática, ou que simplesmente auxiliem os usuários a fazê-las, são extremamente bem-vindas.

O presente trabalho trata de um problema de categorização de textos, que em sua forma mais simples, consiste na classificação binária (*single-label*), onde é possível classificar os documentos em apenas uma de duas categorias. Porém não há impedimentos para que classificações do tipo *multilabel*, onde um mesmo documento pode ser classificado em mais de uma categoria temática, sejam realizadas.

Na maioria dos trabalhos sobre categorização automática de textos, o problema *multi-label* de classificação nas categorias $c_1, \ldots, c_m$ é considerado como m problemas independentes de classificação binária em $\{c_k, \overline{c}_k\}$ para $k = 1, \ldots, m$, onde $\overline{c}_k$ refere-se ao conjunto de todas as categorias, excluída a categoria c_k. Desta forma, m classificadores são construídos, sendo um por categoria. Assim, cada problema de classificação deve ser capaz de determinar se um documento pertence ou não a uma referida categoria.

Em classificação binária, a classe c_k (classe de interesse) é chamada de classe positiva e os documentos desta classe são denominados documentos relevantes ou exemplos positivos. Já a classe $\overline{c}_k$ (demais classes) é chamada de classe negativa e, os seus documentos são denominados como irrelevantes ou exemplos negativos [26].

2.1 Tipos de Abordagens de Dados

Antes do início das atividades de mineração de textos é importante que se saiba como as informações são tratadas em um texto. Duas formas de abordagem podem ser utilizadas: a análise semântica que é baseada na funcionalidade dos termos nos textos e a análise estatística que é baseada na freqüência em que os termos aparecem nos textos (Ebecken et al. [27]).

O estudo dessas abordagens tem por objetivo melhorar a qualidade dos resultados no processo de mineração de textos, através do entendimento do funcionamento da linguagem natural ou da importância de determinados termos no texto (através da freqüência), permitindo a produção de melhores resultados. Essas abordagens são descritas nas seções seguintes.

2.1.1 Análise Semântica

Na análise semântica, as informações textuais são trabalhadas conforme a seqüência em que os termos aparecem no contexto da frase, identificando a correta função de cada termo. Com isso é possível identificar a real importância de cada termo em determinados contextos, possibilitando um ganho na qualidade dos resultados produzidos. Conforme Ebecken et al. [27], o "emprego desse tipo de análise justifica-se pela melhoria em qualidade da mineração de textos quando incrementado de um processamento lingüístico mais complexo".

Na Análise Semântica utiliza-se fundamentos e técnicas baseadas no processamento de linguagem natural. O termo *linguagem natural* nada mais é que o estudo detalhado das linguagens faladas pelos seres humanos a fim de criarem padrões e/ou algoritmos computacionais capazes de simular a capacidade de formulação de textos conforme o ser humano. Para a compreensão da linguagem natural é importante que se entenda os principais tipos de conhecimentos existentes. Dentre eles estão: o conhecimento morfológico, sintático, semântico e pragmático [28].

Desta maneira, na análise semântica dos textos procura-se identificar a importância das palavras dentro da estrutura da oração. A partir deste conhecimento analítico é possível a criação de modelos baseados em regras (criados manualmente por especialistas), para a aplicação em certos sistemas inteligentes (Ebecken et al. [27]).

2.1.2 Análise Estatística

Na Análise Estatística a importância dos termos está diretamente ligada à quantidade de vezes em que eles aparecem nos textos. Este tipo de análise permite, independente do idioma, a criação de modelos através do aprendizado estatístico, capazes de fornecer soluções computacionais suficientemente precisas para tarefas em que a aplicação de mão-de-obra humana é inviável devido à complexidade da tarefa ou do grande volume de dados (Ebecken et al. [27]).

O processo de aprendizado estatístico ou estimativa de dados segue os seguintes passos, definidos por Ebecken et al. [27]:

(i) Codificação dos Dados: é feita a escolha de um modelo de codificação, indicado ou não por um especialista, capaz de identificar características relevantes dentro de um texto e descartar as irrelevantes, buscando sempre manter as principais propriedades dos dados;

(ii) Estimativa dos Dados: após a escolha do modelo de codificação é definido um algoritmo de aprendizado para ser aplicado, com o objetivo de se obter um modelo de estimativa mais adequado para os dados;

(iii) Modelos de Representação de Documentos: o principal modelo de representação de documentos utilizado na tarefa de mineração de textos é conhecido por *bag of words*. Neste tipo de codificação, considera-se uma coleção de documentos como uma espécie de *container* de palavras, ignorando a ordem das palavras dentro do texto assim como os caracteres de pontuação ou estrutura. O número de vezes que uma palavra aparece é mantido. Desta maneira, é possível obter um resumo de todas as informações expressas por um documento.

O conhecimento das técnicas das Análises Semântica e Estatística permitem ao especialista determinar quais as melhores metodologias a serem aplicadas ao seu domínio do problema, de maneira a obter resultados melhores.

2.2 Etapas do Processo de Mineração de Textos

O processo de mineração de textos é composto de 6 etapas. Um esquema ilustrativo deste processo é mostrado na figura 2.1.

Nas subseções a seguir são feitas considerações sobre todas as etapas do processo de mineração de textos.

2.2.1 Coleta de Documentos

A primeira etapa do processo é a coleta de documentos, que consiste na busca de documentos relevantes ao domínio da aplicação do conhecimento a ser extraído. Os documentos

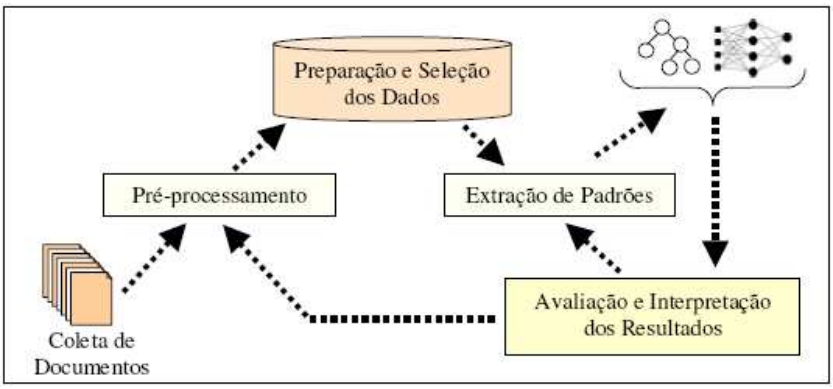


Figura 2.1: Etapas do processo de mineração de textos. Fonte: da Silva [29]

podem ser obtidos de diversas maneiras e em diferentes formatos, como na Internet, livros, periódicos etc. No entanto, segundo Fayyad et al. [30], o sucesso da atividade de coleta de documentos, depende, em parte, da intervenção de um especialista. Segundo os autores, o especialista não só fornece conhecimento sobre o domínio, como também apóia a tarefa de encontrar os objetivos almejados.

Após a coleta dos documentos, é necessário formatá-los, de maneira que os mesmos possam ser manipulados pelos algoritmos de mineração de textos. Esta segunda etapa denomina-se pré-processamento.

2.2.2 Pré-processamento

Após a etapa de coleta de documentos ter sido realizada, é preciso aplicar o pré-processamento. A maior parte do tempo total no processo de extração do conhecimento é despendido nesta etapa, que consiste nas seguintes fases:

(i) Análise léxica;

(ii) Eliminação de termos considerados irrelevantes ou *stopwords* [31–33];

(iii) Normalização morfológica dos termos ou *stemming*.

2.2.2.1 Análise Léxica

Essa fase nada mais é do que a aplicação de um *parser* (analisador léxico) que identifique as palavras presentes nos documentos, ignorando os símbolos e caracteres de controle de arquivo ou de formatação, conforme ilustrado na figura 2.2.

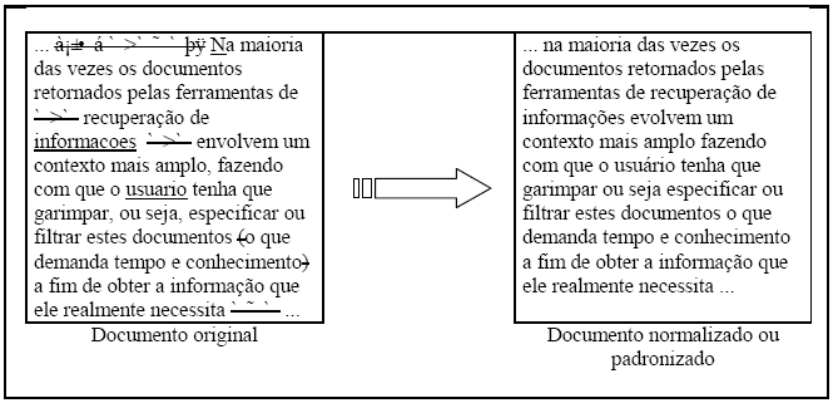


Figura 2.2: Identificação de Termos Válidos. Fonte: Passarin [34]

Pode-se utilizar um dicionário a fim de fazer a validação das seqüências de caracteres identificadas para validar sua existência e corrigir possíveis erros ortográficos (*dictionary lookup*) (Salton e MacGill [33]). Um *thesaurus* ou um dicionário de sinônimos pode auxiliar na normalização do vocabulário, caso deseje-se trabalhar com um vocabulário controlado.

Diversas técnicas adicionais de padronização podem ser aplicadas, ver por exemplo [35] e [36]:

(i) Conversão de todos os caracteres para a forma maiúscula (ou minúscula), também chamado de *case folding*;

(ii) A substituição de múltiplos espaços e tabulações por um único espaço;

(iii) A padronização de datas e números;

(iv) A eliminação de hífens.

Caso quaisquer destas técnicas sejam adotadas, também devem ser implementadas em cima da consulta do usuário.

Por outro lado, a utilização de uma técnica de padronização pode trazer algumas desvantagens. Se, por exemplo, a transformação de caracteres maiúsculos para minúsculos for adotada, não é possível diferenciar substantivos próprios de comuns nas buscas, o que pode ser ruim em algumas situações.

2.2.2.2 *Stopwords*

Quando documentos textuais são examinados, elementos como conjunções, preposições e artigos, são considerados elementos com menos relevância e, por isso são removidos.

Salton e MacGill [33] preconizam que uma das grandes vantagens da remoção dos elementos irrelevantes é a redução de 40 a 50% do tamanho dos documentos.

Stopwords ou *stoplist* são listas de termos irrelevantes que contém palavras que ocorrem com determinada freqüência em textos. Por serem palavras muito comuns, sua presença em pouco contribui para a determinação do conteúdo semântico do documento. Logo, podem ser removidas. Uma *stoplist* bem elaborada permite a eliminação de termos não relevantes, tornando mais eficiente o resultado obtido. A figura 2.3 ilustra como este procedimento é realizado na prática.

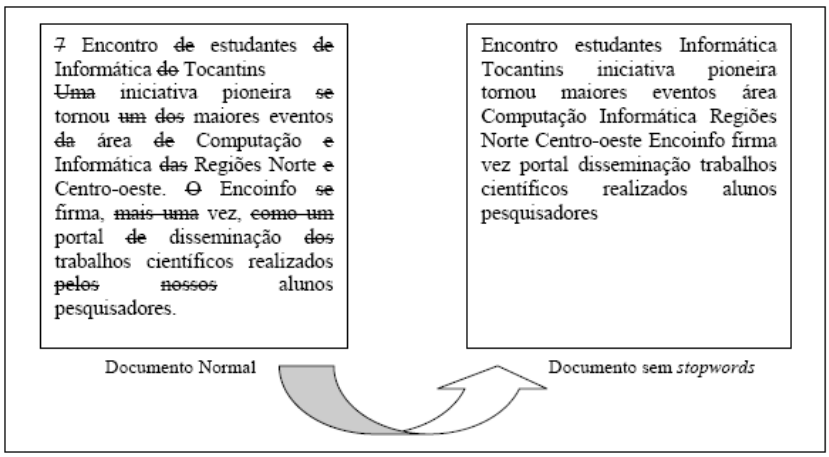


Figura 2.3: Remoção de *Stopwords*. Fonte: Passarin [34]

Existem várias listas de termos irrelevantes. Uma das mais conhecidas para a língua inglesa é a recomendada pelo Laboratório de Recuperação de Informações da Universidade de *Massachusets*, em *Amherst*, composta por 266 termos [37].

Foi realizada uma pesquisa na literatura sobre listas de *stopwords* para a Língua Portuguesa, porém não foram encontradas listas consistentes e completas.

2.2.2.3 *Stems*

Na área de Processamento de Linguagem Natural (PLN), existe uma categoria de técnicas denominada normalização de variações lingüísticas, que permite simplificar palavras, transformando-as em elementos mais simples.

Stemming (radicalização) é o processo de combinar as formas diferentes de uma palavra em uma representação comum, o radical (*stem*) [38]. Tem como objetivo remover prefixos e sufixos, permitindo a recuperação de variações sintáticas das palavras. Um *stem* é a parte que resta de um termo quando são retirados seus afixos. Na figura 2.4 é ilustrada a aplicação

do processo de *stemming*.

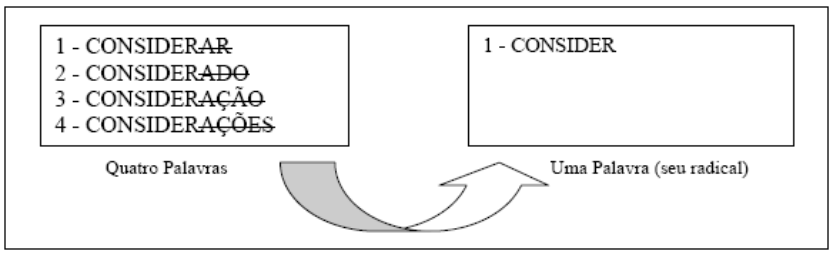


Figura 2.4: Técnica de *Stemming*. Fonte: Passarin [34]

Os algoritmos de *stemming* mais conhecidos são os algoritmos de Porter [39], Lovins [40] e Paice [41] que removem sufixos da língua inglesa.

Tradicionalmente utiliza-se o radicalizador de Porter [39], desenvolvido especificamente para a língua inglesa. Como tal técnica não se adapta bem à língua portuguesa, optou-se pela implementação do algoritmo *Portuguese Stemmer*, proposto por Orengo e Huyck [38]. Apesar destes algoritmos serem utilizados com o objetivo de otimizar o desempenho dos processos de recuperação da informação (RI) e mineração de textos, alguns estudos demonstram que a redução de afixos em uma coleção de documentos não apresenta uma melhora significativa no desempenho da tarefa de mineração aplicada [42, 43], porém somente uma redução no espaço de representação dos dados.

É importante ressaltar que o foco das atividades de pré-processamento é a redução do número de termos do problema, tentando-se manter somente as palavras (*stems*) que concentram a carga semântica do texto.

Na presente dissertação é utilizado o algoritmo de Orengo e Huyck [38].

2.2.3 Preparação dos Dados

Esta etapa envolve a identificação e seleção, nos dados pré-processados, dos termos que melhor expressam o conteúdo de textos, ou seja, toda e qualquer informação que não refletir nenhuma idéia considerada importante deve ser desconsiderada.

Desta maneira, esta seleção reduz a quantidade de termos e, por conseguinte, a dimensão dos vetores que representam os documentos. A redução da dimensionalidade implica na redução da utilização de memória, bem como na redução da carga de processamento. Além disso, reduz a possibilidade de "*overfitting*", fenômeno que ocorre quando o classificador é ajustado de maneira muito específica para o conjunto de treinamento, implicando em uma

baixa performance na classificação de documentos não conhecidos pelo classificador [44].

A preparação dos dados é a primeira fase do processo de criação indutiva de classificadores de texto. Deve-se entender por indução a possibilidade de obtenção de conclusões genéricas sobre um conjunto particular de exemplos (conjunto de treinamento). O raciocínio indutivo é utilizado para generalizar conceitos, aproximar funções e descobrir novas funções através de exemplos fornecidos.

Com o objetivo de reduzir a quantidade de termos, algumas etapas são realizadas, visando um melhor desempenho do sistema na manipulação dos documentos. Na figura 2.5 é ilustrada a seqüencia de etapas a serem cumpridas.

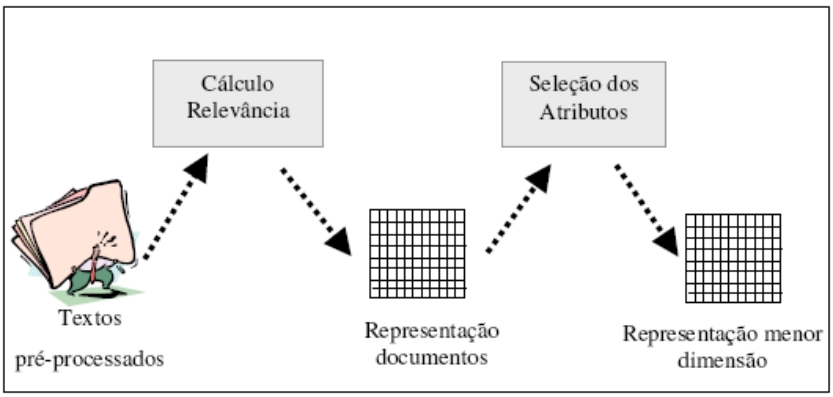


Figura 2.5: Etapas da Preparação de Dados. Fonte: da Silva [29]

2.2.3.1 Cálculo de Relevância

De acordo com Meadow et al. [45], muitos termos não possuem significado semântico suficiente para descrever o assunto de um texto. Desta maneira, é interessante que se use uma medida para calcular a relevância dos termos nos documentos.

Os métodos mais utilizados para o cálculo de relevância são os baseados na freqüência de termos [46, 47], são eles:

(i) Freqüência absoluta;

(ii) Freqüência relativa;

(iii) Freqüência inversa de documentos.

A *freqüência absoluta*, mais conhecida como *Term Frequency*, indica o número de ocorrências de uma palavra em um documento. É normalmente denominada de *tf*. Para cada

documento, a palavra pode possuir uma freqüência diferente, dependendo do seu número de ocorrências. É considerada a mais simples das medidas de peso, porém seu uso não é indicado, por não ser capaz de fazer distinção entre os termos que ocorrem em poucos documentos e os termos que ocorrem com alta freqüência em muitos documentos. Este é um fator importante, visto que termos que ocorrem em muitos documentos não possuem poder de distinção. Além disso, a freqüência absoluta não leva em consideração a quantidade de termos existentes no documento. Por exemplo, se em um documento pequeno, um determinado termo que ocorre muitas vezes, o mesmo pode ser menos representativo do que outro que ocorre poucas vezes em um documento extenso. O contrário também pode ocorrer.

A *freqüência relativa* consiste na freqüência absoluta (*tf*) normalizada pelo tamanho do documento (número de palavras nele contidas). A normalização é feita dividindo-se o número de ocorrências da palavra pelo número total de palavras presentes no documento. O resultado é um número entre 0 e 1, que indica a importância relativa de uma palavra em um documento. Assim, todos os documentos acabam por "ter o mesmo tamanho", independente do seu número absoluto de palavras. Essa medida é também conhecida por freqüência normalizada, ou *normalized term frequency*. Sem essa normalização, os documentos grandes e pequenos seriam representados por valores em escalas diferentes. Com isso os documentos maiores possuem melhores chances de serem recuperados, uma vez que receberão valores maiores no cálculo de similaridades [47], e vice-versa.

Apesar de muito utilizada, esta medida determina que palavras que aparecem em muitos documentos têm a mesma importância de palavras que estão em poucos documentos, possuindo um maior poder de discriminação [48].

A freqüência relativa de um termo x em um documento qualquer, pode ser calculada dividindo-se a freqüência do termo tf pelo número total de termos no documento N, isto é,

$$Frelx = \frac{tf(x)}{N} \tag{2.1}$$

A freqüência inversa de documentos, conhecida por idf, é uma medida que leva em conta tanto a freqüência absoluta quanto a freqüência de documentos, que é o número de documentos em que a palavra aparece. Assim, palavras que aparecem em muitos documentos são penalizadas pelo mesmo motivo das *stopwords*, ou seja, pela pouca discriminação que oferecem.

Com esse cálculo é possível indicar a quantidade de documentos em que um termo ocorre, baseando-se no cálculo da freqüência do termo e na remoção do espaço de características dos termos, cuja freqüência de documentos é inferior a um determinado limiar. Por exemplo, dados os termos mais freqüentes, são recuperados os documentos em que o termo aparece no mínimo x vezes. Essa técnica é considerada menos complexa para a redução de termos, como também é facilmente escalável para grandes massas de textos, com uma complexidade computacional aproximadamente linear em relação ao número de documentos.

Com base nos cálculos de freqüência dos termos e freqüência dos documentos é possível obter a *freqüência do termo - freqüência inversa de documentos*, mais conhecida como *tf-idf*. Através desta medida pode-se aumentar a importância de termos que aparecem em poucos documentos e diminuir a importância de termos que aparecem em muitos documentos. Segundo Salton e MacGill [33], os termos de baixa freqüência de documentos, são, em geral, mais discriminantes.

Uma das maneiras mais utilizadas para se identificar o peso do termo é a aplicação da fórmula preconizada por Salton e MacGill [33], dada por:

$$Tf - Idf = \frac{Freq_{td}}{DocFreq_t} \tag{2.2}$$

sendo que $Freq_{td}$ é o número de vezes em que o termo t aparece no documento d e $DocFreq_t$ corresponde ao número de documentos em que o termo t aparece.

2.2.3.2 Representação de Documentos

Baseado na relevância das palavras nos textos, é possível realizar a codificação dos dados e escolher um modelo de representação de documentos [48]. A codificação dos termos e a representação dos documentos consistem na escolha de um modelo que possa ser compreendido pelos algoritmos de mineração de textos a serem utilizados. Geralmente a codificação dos documentos deve seguir as indicações de um especialista, ou algum critério escolhido que reflita propriedades interessantes dos dados em relação à finalidade da modelagem. A codificação mais freqüentemente utilizada é a sugerida na técnica *bag of words* [49], na qual cada documento é representado por um vetor composto de termos presentes no documento. Esta codificação pode ser considerada uma simplificação das informações expressas por um documento, não fornecendo, portanto, uma descrição fiel do conteúdo.

Alguns modelos de representação de documentos foram desenvolvidos na área de Recuperação de Informação (RI). Os mais utilizados são o modelo Booleano e o modelo de Espaço de Vetores, propostos por Baeza-Yates e Ribeiro-Neto [50].

A abordagem booleana verifica a presença ou ausência da palavra no documento, determinando pesos a esses termos de forma binária, isto é, 0 ou 1. Pelo fato do modelo ser binário, a freqüência de um termo não tem efeito. Uma grande vantagem deste modelo é a sua simplicidade e a necessidade de pouco espaço de armazenamento. É considerada a mais clássica das representações utilizadas em RI. Alguns mecanismos de busca na *web* são baseados neste modelo, como Excite, Alta Vista e Lycos [51].

Os problemas relacionados ao modelo booleano são, no entanto, bem conhecidos [52]. A formulação de uma consulta adequada é difícil, isto é, a seleção dos termos para a consulta é difícil, principalmente se o domínio não for bem conhecido. O tamanho da saída não pode ser controlado. O conjunto resultante tanto pode conter nenhum como milhares de itens. Além disso, sem um grau de comparação parcial, não se pode saber o que foi deixado de fora da definição da consulta. Uma vez que não há um grau de comparação, não é possível ordenar os resultados de acordo com a relevância.

O modelo de espaço de vetores, conhecido com VSM, visa contornar as limitações do modelo booleano, com a utilização de pesos e, desta maneira, permitir uma similaridade parcial entre os documentos e as palavras.

No modelo de espaço de vetores, como na maioria dos modelos de recuperação de informação, a expressão de igualdade entre termos é representada através do que é chamado de similaridade. Então, adota-se este termo como forma de representação do número de propriedades (ou características) que determinados objetos (documentos ou termos) possuem em comum, assim como o número de propriedades incomuns entre objetos [53].

Neste modelo, os documentos são representados por vetores de termos, ou seja, a organização e representação dos documentos textuais neste modelo são feitas através de vetores de termos-índices que descrevem as propriedades de cada documento em particular. Estes vetores são representados pela forma $Di = (t_1, t_2, t_3, ..., t_n)$, em que Di é o *i-ésimo* documento da coleção, e t_n o *n-ésimo* termo do documento [53], ou seja, para cada documento da coleção existem n termos-índices que os representa.

Para cada termo armazenado no vetor existe um valor associado, que informa o grau de importância deste termo no documento (também denominado como peso). Portanto, cada

documento textual possui um vetor relacionado que é representado por pares de elementos na forma ($termo_1$, $peso_1$), ($termo_2$, $peso_2$), ..., ($termo_n$, $peso_n$). A importância de um termo é normalmente calculada através da freqüência em que o mesmo aparece no documento, ou seja, se o "$termo_1$" aparecer 10 (dez) vezes dentro de um determinado documento, este será o valor da sua importância.

É importante ressaltar que neste modelo as consultas feitas pelos usuários também são representadas por vetores de termos-índices, com o formato idêntico ao dos vetores dos documentos[53]. Portanto, a forma de se calcular a similaridade sempre será a mesma, seja entre documentos ou através de consultas.

Como todos os documentos e consultas são representados como vetores, os documentos podem ser dispostos em um espaço euclidiano de dimensão n, em que n é o número de termos, é cada termo é um eixo deste espaço euclidiano. Esta representação dimensional é ilustrada na figura 2.6.

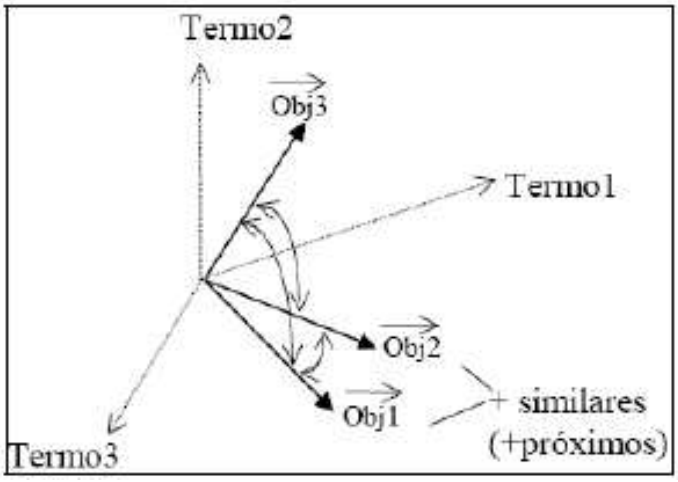


Figura 2.6: Modelo de Espaço de Vetores. Fonte: Wives [54]

Conforme a figura 2.6, cada termo (por exemplo, Termo1) é considerado como uma dimensão. O valor do documento (por exemplo, Obj1) em relação a cada eixo varia entre 0 (não similaridade ou irrelevância) e 1 (totalmente relevante ou similar).

Uma das formas de se calcular a similaridade entre documentos é testar o valor do ângulo existente entre os mesmos (por exemplo, o ângulo entre Obj1 e Obj2). Desta maneira, quanto menor for este valor, maior será o grau de similaridade. Salton e MacGill [33] propuseram uma função para calcular a similaridade por cossenos, dada por:

$$Similaridade(Q, D) = \frac{\sum_{k=1}^{n} W_{qk} W_{dk}}{\sqrt{\sum_{k=1}^{n} (W_{qk})^2 \sum_{k=1}^{n} (W_{dk})^2}} \quad (2.3)$$

onde Q é o vetor de termos-índices da consulta, D é o vetor de termos-índices do documento, W_{qk} são os pesos dos termos-índices da consulta e W_{dk} são os pesos dos termos-índices do documento.

Após a finalização do processo de cálculo de similaridades, é possível a criação de listas de resultados de todos os documentos e seus respectivos graus de relevância, ordenadas de forma decrescente. Desta maneira, o processo de consulta está finalizado e o resultado é uma lista dos documentos que satisfizeram as necessidades da consulta, conforme a idéia central do processo de recuperação de informações.

Um problema óbvio com o modelo de espaço de vetores é a sua dimensão elevada: o número de palavras diferentes em uma coleção de documentos facilmente atinge centenas de milhares. Outro problema conhecido é composto por variações de estilo de escrita, erros de grafia etc [52].

Na figura 2.7 é ilustrada geometricamente a representação dos documentos *doc1*, *doc2*, *doc3*, *doc4* e *doc5* em relação aos termos *linguagem*, *natural* e *processamento*.

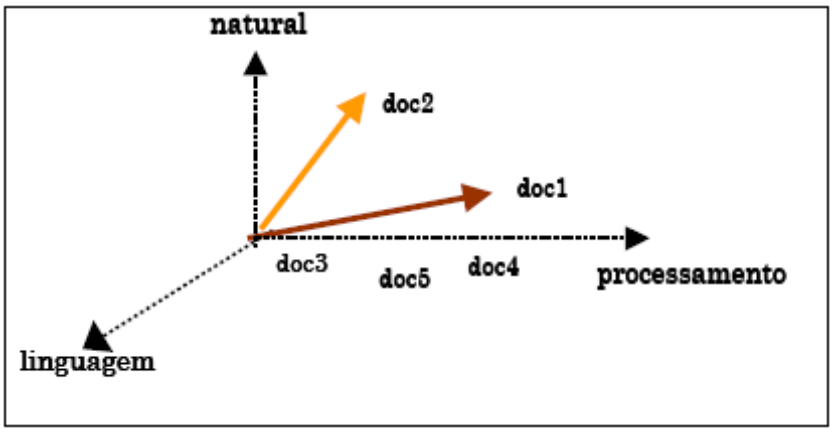


Figura 2.7: Modelo de Espaço de Vetores. Fonte: da Silva [29]

Pode-se observar na figura que o termo "processamento" está associado ao vetor *doc1* e que o termo "natural" está associado ao vetor *doc2*.

2.2.3.3 Seleção de Atributos

O desempenho de uma tarefa de mineração de textos pode tornar-se computacionalmente inviável se todos os termos de um documento forem considerados para a geração de sua

representação.

Assim, é necessário que se faça a seleção dos atributos. Esta consiste na eliminação de palavras que não são representativas, ou então na combinação de mais de uma palavra em um único atributo, de maneira a diminuir o número de elementos que compõem os vetores dos documentos. Porém algumas aplicações são influenciadas pelos termos de menor importância (por exemplo, agrupamento, classificação e sumarização) [55]. É necessário que o usuário ou o especialista decida sobre quais termos são relevantes ou não para o experimento. Os métodos de seleção mais comumente utilizados são:

(i) Filtragem baseada no peso do termo;

(ii) Seleção baseada no peso do termo;

(iii) Seleção por indexação semântica latente;

(iv) Seleção por linguagem natural.

A *filtragem baseada no peso do termo*, consiste em filtrar os termos inferiores a um limiar (*threshold*) estabelecido pelo usuário ou pela aplicação [55]. Porém, mesmo depois de filtrados, o número de termos selecionados ainda pode ser alto. O problema da alta dimensionalidade pode ser resolvido aplicando-se a seleção dos n termos mais relevantes. Esta técnica é denominada de *seleção do peso do termo* ou *truncagem*. Ela estabelece um número máximo de características a serem utilizadas para caracterizar um documento. Todas as outras são descartadas.

Um dos problemas deste método de seleção é a determinação da quantidade mínima de termos necessária para melhorar as descrições dos documentos, de maneira que suas características mais relevantes não sejam perdidas no processo.

O método de seleção por *indexação semântica latente* tem por objetivo reduzir a dimensão do espaço de vetores, fazendo uma análise da estrutura co-relacional de termos na coleção de documentos. Essa redução é realizada identificando-se os termos mais similares. Assim que estiverem identificados, eles são aproximados por um processo matemático de rotação, de maneira que os termos mais similares (próximos) fiquem no mesmo eixo. Sinônimos e outros termos de forte correlação, em alguns casos, são colocados no mesmo eixo, minimizando assim, problemas relacionados à diferença de vocabulário.

Também é possível aplicar algumas técnicas de análise de linguagem natural [33] para identificar os termos mais relevantes de um documento. Técnicas como essas são denominadas de *seleção por linguagem natural*. Elas incluem a análise sintática e semântica dos documentos.

Com uma gramática bem definida para um domínio específico, também conhecida como léxico [56], é possível realizar uma análise sintática em orações com baixa complexidade. Os objetos que compõem uma oração costumam ter posições bem definidas, de maneira que é possível influenciar o peso dos termos encontrados, a fim de torná-los mais ou menos relevantes. Porém, este tipo de técnica exige uma base de conhecimento que contenha todas as combinações sintáticas possíveis, ou seja, uma gramática.

2.2.4 Extração de Padrões

Inicialmente, a motivação principal na Extração de Padrões ou Extração de Informações (EI) era a de povoar automaticamente bases de dados [57]. Entretanto, sistemas de extração de informações também permitem melhorar o desempenho de sistemas de recuperação de informações através da integração e sintetização da informação, evitando desta maneira, a ocorrência de redundâncias em textos que tratam do mesmo assunto.

Segundo Wives e Loh [58], a EI foi uma evolução natural da área de recuperação de informações. Constantemente, mais e mais informações surgem nos meios eletrônicos e uma porcentagem significativa das mesmas é composta de informações que, de alguma maneira, podem ser estruturadas ou interrelacionadas. Assim, ao invés de encontrar textos que contenham informações e permitir ao usuário procurar o que lhe interessa, esta nova área tem como objetivo encontrar as informações dentro dos textos e tratá-las de forma a apresentar algum tipo de conhecimento novo e útil para o usuário. A idéia é aproveitar todo o conhecimento humano que há em textos escritos e, mesmo que tal conhecimento novo não seja resposta direta às indagações do usuário, deve-se processá-lo sob a premissa de que ele deve contribuir para a satisfação das necessidades de informação do mesmo.

Nas palavras de Wives e Loh [58] “as aplicações de sistemas de descoberta de conhecimento em textos são inúmeras. Qualquer domínio que utilize intensivamente textos não-estruturados (documentos escritos), tais como as áreas jurídicas, policiais, cartórios e órgãos de registros, empresas em geral etc, podem beneficiar-se destes sistemas.”

Do ponto de vista das técnicas utilizadas, a EI pode ser vista como qualquer método que filtre informações de um grande volume de texto. Grishman [25] define EI como "a identificação de instâncias de uma classe particular de eventos ou relacionamentos num texto em linguagem natural, e a extração de argumentos relevantes do evento ou do relacionamento". O autor conclui que a extração de informações envolve a criação de uma representação estruturada da informação selecionada e extraída do texto.

Tipicamente, a extração de informação envolve a identificação de padrões que representam um contexto chave dentro do texto. Além disso, a EI utiliza um conjunto de filtros que, junto com os padrões, representam, de forma estruturada, a informação contida em cada texto, possibilitando a atualização de uma base de dados ou a melhora de uma recuperação de informações posterior [57].

Dentre as diversas técnicas implementadas, pode-se destacar:

(i) Identificação de Tópicos: um sistema de identificação de tópicos mantém os perfis do usuário e, baseado nos documentos que o usuário já manipulou, consegue predizer outros documentos de interesse do mesmo.

A identificação de tópicos pode ser usada para, por exemplo, alertar uma empresa toda vez que o nome de uma concorrente surgir em um jornal, possibilitando desta maneira, uma monitoração das atividades da concorrência, de modo a auxiliar na tomada de decisões;

(ii) Sumarização: segundo Spark-Jones e Willet [59], sumarização é a abstração das partes mais importantes do conteúdo de um texto.

A análise do texto é feita sobre sua organização (seções, parágrafos, títulos, subtítulos), sobre as sentenças que o compõem (análise morfológica e sintática com uso de um dicionário), sobre a estrutura do texto (conectivos lógicos, expressões idiomáticas entre parágrafos e frases) e com a extração de papéis ou funções (roles) semânticas por *tags* (para cada função, são definidos *tags* específicos; por exemplo, para achar tópicos, procurar pelo *tag* "... foi desenvolvido").

A principal atividade da sumarização é reduzir o tamanho e os detalhes do documento, enquanto preserva os pontos principais e mais abrangentes.

A sumarização tem a função de reconhecer em um texto, o que é relevante e o que

pode ser descartado, para compor um sumário. Esse é também um dos pontos mais problemáticos e sujeitos a controvérsias. A importância de uma sentença ou trecho de um texto pode depender de vários fatores: (1) dos objetivos do autor do sumário; (2) dos objetivos ou interesse de seus possíveis leitores; (3) da importância relativa (e subjetiva) que o próprio autor (ou leitor) atribui às informações textuais, fazendo com que mesmo a estruturação do sumário esteja sujeita à complexidade de se determinar forma e conteúdo a partir do fonte correspondente. Na figura 2.8 é ilustrado o processo de sumarização.

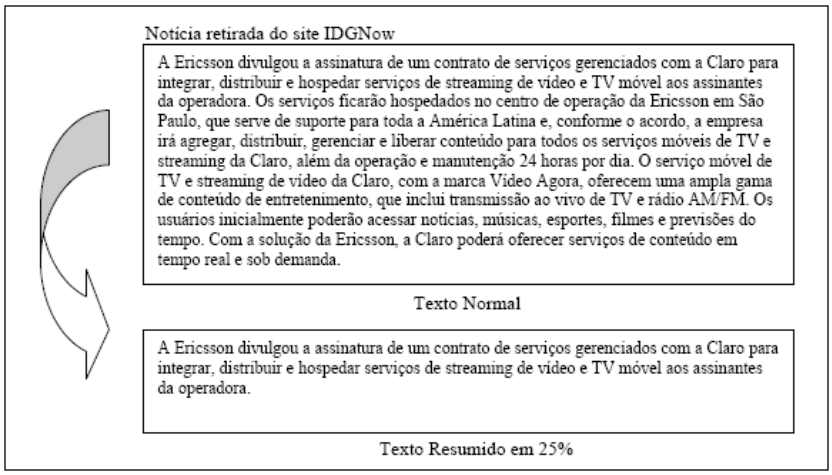


Figura 2.8: Processo de Sumarização. Fonte: Passarin [34]

(iii) Clusterização: clusterização ou *Clustering* é um processo de particionar um conjunto de objetos de dados em um conjunto de subclasses significativas, chamadas *clusters*. Formalmente, consideremos uma coleção de n objetos descritos por um conjunto de p atributos. O processo de *clustering* visa obter uma divisão útil de n objetos em um número de clusters. Um *cluster* é uma coleção de objetos de dados que são similares uns aos outros, baseados em seus valores de atributos, e assim podem ser tratados coletivamente como um grupo. O processo de clusterização é muito útil no entendimento da distribuição de um conjunto de dados. Um algoritmo de *clustering* busca grupos naturais nos dados baseados em similaridade de atributos. Na figura 2.9 é ilustrado o processo de clusterização.

(iv) Categorização: esta técnica visa identificar os tópicos principais de um documento, e associá-lo a uma ou mais categorias pré-definidas [55]. Muitas das técnicas de extração de padrões utilizadas em categorização de documentos são similares às utilizadas em

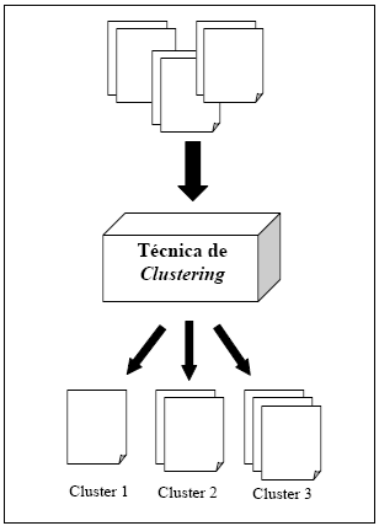


Figura 2.9: Processo de Clusterização. Fonte: Ebecken et al. [27]

mineração de dados [10, 60]. Considerações adicionais são feitas na seção 2.2.6.

2.2.5 Avaliação e Interpretação de Resultados

Sempre que um procedimento de mineração é executado, é necessário avaliar o resultado retornado em função das informações relevantes ou irrelevantes obtidas [61]. Nesta fase, os resultados podem ser analisados com a finalidade de constatar se o objetivo foi alcançado. As medidas de avaliação do desempenho de um sistema são originárias da área de RI e baseadas na idéia de relevância, ou seja, se um documento atender a necessidade de informação do usuário, ele é considerado relevante à solicitação do mesmo. Infelizmente, não é possível evitar o retorno de dados indesejados, porém é possível avaliar se o retorno da busca foi satisfatório, através das métricas: "precisão", "cobertura" e "medida F" [31–33] e podem ser obtidas através das seguintes relações, discutidas em [46]:

(i) **Abrangência**, Recordação ou *Recall*: consiste na avaliação da habilidade do sistema em recuperar os documentos mais relevantes do usuário [62], medindo a quantidade de itens recuperados, dentre os relevantes na base de dados. A abrangência é dada por:

$$abrangência = \frac{n_recuperados_relevantes}{n_relevantes} \tag{2.4}$$

(ii) **Precisão** ou *Precision*: avalia a habilidade do sistema manter os documentos irrelevantes fora do resultado de uma consulta [62]. A precisão é definida pela razão entre

a quantidade de itens recuperados dentre os relevantes e o número total de itens recuperados, isto é,

$$precisão = \frac{n_recuperados_relevantes}{n_total_recuperados} \tag{2.5}$$

Contemplar separadamente as medidas de *precisão* e *abrangência* pode levar a uma avaliação equivocada do sistema, visto que, geralmente, quando se aumenta a *precisão*, a *abrangência* do sistema é diminuída [46].

(iii) *medida F* ou *F-Measure*: combina os valores de *precisão* e *abrangência*, de maneira a se obter o desempenho geral do sistema, permitindo um balanceamento entre os dois valores. É definida por:

$$F = \frac{(\beta^2 + 1)PC}{\beta^2(P + C)} \tag{2.6}$$

onde β é o parâmetro que permite a atribuição de diferentes pesos para as medidas de *precisão (P)* e *abrangência (C)*, sendo 1 o valor geralmente adotado. O valor F é maximizado quando P e C são iguais ou muito próximos, sendo F o ponto de equilíbrio do sistema.

2.2.6 O Processo de Categorização

A categorização consiste na identificação do tema principal de um documento [55]. Categorizar é classificar automaticamente documentos em relação a um conjunto de categorias ou matérias previamente conhecidas. No paradigma biblioteconômico, classificar é colocar um documento numa classe, dentro de um esquema, ou agrupar os documentos com características comuns, em função dos seus conteúdos.

Para criar categorias é necessário que cada uma delas possa ser representada por um termo ou conjunto de termos que suportam o significado do conceito associado. Metodologicamente, o problema reside na identificação de um termo ou conjunto de termos cujo significado semântico possa definir o conceito de uma determinada categoria. A indexação tem duas etapas fundamentais: a análise e a representação. Através da análise identificam-se os conceitos que constituem unidades de pensamento do conteúdo temático de um documento,

expressas em linguagem natural. No entanto, conceito e termo não se identificam, na medida em que um conceito é uma idéia ou noção, a representação intelectual de um objeto, enquanto que o termo é a sua representação formal e o seu suporte visível. Por isso é necessário estabelecer, com rigor, as regras de associação dos termos aos conceitos. Para essa finalidade foram criadas as linguagens controladas de indexação. Uma vez selecionado ou criado o instrumento de linguagem controlada, os termos tidos como representativos do conteúdo do texto são expressos em linguagem de indexação, ficando resolvidas algumas das ambigüidades da linguagem natural. Esta representação pode ser vocabular – simples ou composta – ou simbólica. Finalmente os termos de indexação são integrados na respectiva categoria. O termo de indexação tem como característica principal a sua funcionalidade: atua na pesquisa como ponto de acesso à informação, como uma porta de entrada, por assunto, num sistema informativo. Este sistema prevê a interação do usuário para classificar a categoria determinada por um conjunto de termos.

Em resumo, a categorização de textos é uma ferramenta utilizada para classificar automaticamente um conjunto de documentos numa ou mais categorias preexistentes, não tendo outra finalidade senão recuperar a informação. Esta técnica é usualmente utilizada, num sentido mais amplo, para filtrar e encaminhar mensagens de correio eletrônico, organizar notícias, resumos e arquivos de publicações periódicas. Na figura 2.10 é ilustrada a representação do processo de categorização.

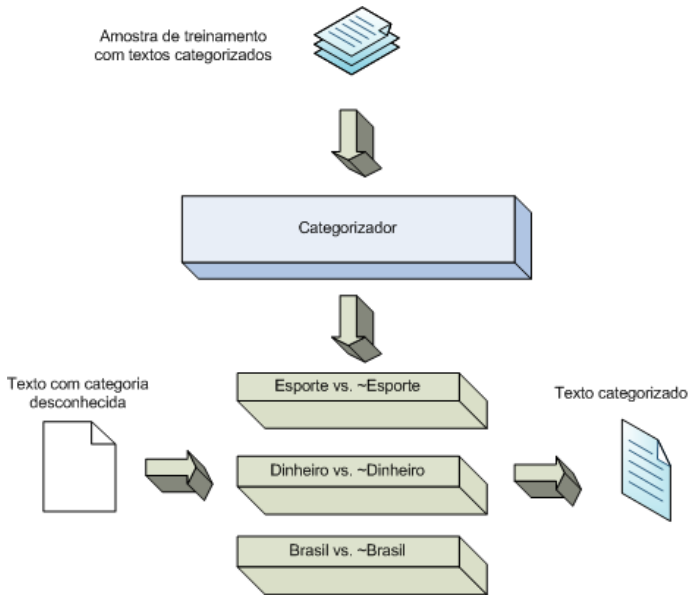


Figura 2.10: Processo de Categorização

No presente trabalho são realizados testes com os categorizadores bayesiano e de ran-

queamento linear propostos por Weiss et al. [63].

2.2.6.1 Algoritmo *Naive-Bayes*

O classificador *Naive Bayes* (NB) é provavelmente o mais utilizado para classificação de documentos. É baseado no teorema de *Bayes* [64, 65], cuja idéia básica é usar a junção das probabilidades das palavras e categorias para estimar as probabilidades das categorias de um novo documento.

Desta maneira, o algoritmo calcula a probabilidade a *posteriori* de um documento pertencer a classes diferentes e o atribui à classe cuja probabilidade a *posteriori* é a mais alta. A probabilidade a *posteriori* é calculada usando-se a regra de *Bayes* e o conjunto de teste é atribuído à classe com a mais alta probabilidade a *posteriori*.

A parte ingênua (*naïve*) do algoritmo NB é a suposição de independência das características da palavra, ou seja, é assumido que o efeito das características de uma palavra cuja probabilidade condicional está associada a uma categoria é independente das características de outras palavras daquela categoria. Apesar desta premissa "ingênua" e simplista, este classificador apresenta um excelente desempenho em várias tarefas de classificação.

Até as primeiras décadas do século XVIII, problemas relacionados à probabilidade de certos eventos, dadas certas condições, estavam bem resolvidos. Por exemplo, dado um número específico de bolas pretas e brancas em uma urna, qual é a probabilidade da bola preta ser sorteada? Tais problemas são denominados de *forward probability*. Porém, logo o problema inverso começou a chamar a atenção dos matemáticos da época: dado que uma ou mais bolas foram sorteadas, o que pode ser dito sobre o número de bolas brancas e pretas na urna? É exatamente este pensamento inverso que busca-se ao treinar um classificador de textos.

Thomas Bayes, um ministro inglês do século XVIII, foi o primeiro a formalizar uma teoria desta natureza, vista como revolucionária no meio científico da época.

Vários experimentos têm sido feitos com esse algoritmo, ver por exemplo [66], apresentando resultados satisfatórios para classificação de padrões, porém outros métodos também se destacam, tendo uma utilização mais direcionada para outros tipos de problemas dentro da mineração de textos [61].

O algoritmo *naive bayes* apresenta as seguintes vantagens:

(i) É de fácil implementação, pois o algoritmo é simples;

(ii) Não requer nenhum procedimento de aprendizagem (as probabilidades são estimadas baseadas nas freqüências de termos);

(iii) Capacidade de aprender através de exemplos;

(iv) O processo de classificação é eficiente, já que as características são independentes das outras.

Por outro lado, as seguintes desvantagens deste algoritmo podem ser destacadas:

(i) Requer o conhecimento inicial de muitas probabilidades;

(ii) O custo computacional é linear, porém está vinculado à quantidade de palavras e de características existentes.

O método *naive bayes*, inicialmente, pode parecer complexo mas, de fato, possui uma estrutura linear, apesar de possuir uma exponencial em sua estrutura. Isto pode ser visto pela definição 2.7.

$$Pr(C|x) = \frac{1}{Pr(x)} exp(\sum_{j} w_j x_j + b) \tag{2.7}$$

onde

$$w_j = ln\frac{Pr(x_j = 1|C)}{Pr(x_j = 0|C)}, b = lnPr(C) + \sum_{j} lnPr(x_j = 0|C). \tag{2.8}$$

sendo C a categoria procurada, x uma palavra que compõe o dicionário, w uma palavra que compõe um documento e b é uma constante.

Na figura 2.11 é mostrado o pseudocódigo do algoritmo *Naive Bayes*. Em [67] é apresentada uma discussão detalhada sobre o algoritmo *Naive Bayes*.

2.2.6.2 Algoritmo de Ranqueamento Linear

De maneira a se obter uma boa performance na atividade de categorização/predição, é muitas vezes necessário a criação de vetores de características com dimensões muito elevadas. Embora muitas dessas características não sejam utilizadas, torna-se difícil para uma

```
1. Para cada classe:
2.    Calcular sua probabilidade
3.    Para cada atributo do documento:
4.       Calcular sua probabilidade para cada classe
5.       Multiplicar todos os valores calculados para esta classe
6. Multiplicar o valor obtido pela probabilidade da classe
```

Figura 2.11: Pseudocódigo do classificador *Naive Bayes*

pessoa dizer quais delas são úteis ou não. Deste modo, os algoritmos de categorização devem ser capazes de manipular um grande conjunto de características e selecionar somente os que forem realmente relevantes. Uma maneira muito utilizada para realizar esta tarefa é pelo uso de algoritmos de ranqueamento linear.

O algoritmo *Naive Bayes*, descrito acima, pode ser considerado um caso especial de um algoritmo de ranqueamento linear. Todavia, sua performance pode ser melhorada significativamente usando-se métodos de treinamento mais sofisticados para se obter o vetor de pesos $w = [w_j]$ e o bias b.

Considere o problema de classificação para duas classes. O método de ranqueamento linear consiste em assinalar com uma pontuação positiva as classes identificadas como positivas e com uma pontuação negativa as classes identificadas como negativas. Na tabela 2.1 é ilustrado um exemplo do uso do conjunto de pesos para determinar uma pontuação para um documento. Na tabela 2.2 é ilustrado um exemplo de cálculo do ranqueamento para um novo documento baseado nos pesos da tabela 2.1.

Tabela 2.1: Pesos das características para o modelo linear

Palavra	Peso
dividendos	0.8741
passos	0.4988
oito	-0.0866
extraordinário	-0.0267
meses	-0.1801
pagamento	0.6141
divisão	0.9050

Tabela 2.2: Cálculo do ranqueamento baseado nos pesos das características

Palavra	Peso
dividendos, pagamento, meses	1.3081

Para todas as palavras que ocorrem no documento, encontra-se seu peso correspondente.

Esses pesos são, enfim, somados para se determinar a pontuação do documento. Matematicamente, este método é uma função de ranqueamento linear, cuja fórmula geral é dada por:

$$Score(D) = \sum_{j} w_j x_j + b = wx + b. \tag{2.9}$$

onde D é o documento e w_j é o peso para a j-ésima palavra no dicionário, b é uma constante, e x_j é um ou zero, dependendo da presença ou ausência da j-ésima palavra no documento.

Métodos de ranqueamento linear são abordagens clássicas para a resolução de problemas de categorização/predição. Geometricamente, este método consiste na geração de uma reta ou hiperplano no espaço. Embora superfícies complexas não possam ser ajustadas por uma reta, freqüentemente é possível criar características não lineares adequadas, de tal modo que a curva no espaço original esteja sobre um hiperplano no espaço aumentado, com as características não lineares adicionais. Desta maneira, a não linearidade pode ser explicitamente capturada com a construção de sofisticadas características não lineares. Uma vantagem dessa abordagem é que o aspecto da modelagem torna-se muito simples, uma vez que o foco está na criação de características úteis e permitir que o algoritmo possa determinar um peso para cada característica criada. Outra vantagem é que este método funciona de maneira muito eficiente com dados esparsos. Isto é muito importante para aplicações de mineração de textos, uma vez que apesar dos vetores de características apresentarem grandes dimensões, eles são, usualmente, muito esparsos.

Sabe-se de diversos *benchmarks* em que a abordagem de ranqueamento linear funciona surpreendentemente bem na aplicação de classificação de textos.

Na figura 2.12 é mostrado o pseudocódigo do algoritmo de Ranqueamento Linear.

```
1.  Para todas as palavras do documento:
2.    Calcular seu peso
3.    Acumular os pesos
```

Figura 2.12: Pseudocódigo do classificador de ranqueamento linear

Em Weiss et al. [63] encontra-se uma discussão mais aprofundada sobre o assunto.

2.3 Ferramentas de Mineração de Textos

As tarefas de mineração de textos já podem contar com vários tipos de ferramentas computacionais, sejam elas gratuitas ou comerciais.

O uso destas ferramentas é crucial para as atividades inerentes ao processo de mineração de textos. A grande variedade destas ferramentas permite que pesquisadores e organizações selecionem as que melhor atendem os seus objetivos. Para um estudo sobre uma grande variedade de ferramentas para mineração de textos ver Gemert [68].

A seguir são descritas algumas destas ferramentas, algumas de suas características e aplicabilidade.

2.3.1 Ferramentas de Domínio Público

2.3.1.1 *Text Mine*

Text Mine, descrito por Konchady [69], é um conjunto de ferramentas de mineração de textos escritas em *Perl*, que é uma linguagem apropriada para o desenvolvimento de scripts para servidores *Web*.

Os requisitos básicos desta ferramenta são: (1) linguagem de programação *Perl*; (2) *Apache Server* como servidor de aplicações; (3) banco de dados *MySql*.

Text Mine disponibiliza as seguintes funcionalidades:

(i) Busca: a busca pode ser feita a partir de URL's, onde o *crawler* visita as páginas, construindo o conjunto de informações necessárias para as outras funcionalidades, ou em documentos locais à máquina;

(ii) Extração de informação: é capaz de identificar entidades no texto como nomes, locais e organizações. Para isso, o usuário deve customizar um dicionário de entidades que serão identificadas;

(iii) *Clustering*: retorna uma coleção de grupos, onde cada grupo é composto por um número variável de documentos similares. Gera uma matriz de similaridades da coleção de documentos e utiliza algoritmo genético para agrupá-los segundo o grau de similaridade existente entre eles;

(iv) *Tracking*: consiste em organizar as mensagens de texto em categorias;

(v) Sumarização: consiste em resumir artigos ou documentos da *Web*, extraindo palavras-chave que determinam o significado do documento.

Esta ferramenta possui versões compatíveis com ambiente *MS Windows* e *Linux*.

2.3.1.2 *TMSK - Text-Miner Software Kit*

O *TMSK* [63] é um pacote de softwares para mineração preditiva de textos. Possui funcionalidades para pré-processar documentos de textos em formato *XML* e disponibiliza implementações para as seguintes tarefas:

(i) Pré-processamento: implementa funcionalidades de tokenização, *stemming*, criação de dicionário e detecção de fim de sentença;

(ii) Predição: possui classificadores baseados em regras de decisão, predição usando *Naïve Bayes* e modelos de ranqueamento linear;

(iii) Recuperação de informação: implementa o conceito de listas invertidas (termos que apontam para os documentos);

(iv) *Clustering*: realiza o agrupamento de documentos usando o algoritmo *k-Means*;

(v) Extração de informações: identificação de entidades nomeadas (NER).

O TMSK funciona em qualquer tipo de *hardware* e é multiplataforma, sendo necessário para o seu funcionamento apenas uma interface de linha de comando e um interpretador Java (*Java Virtual Machine*) na versão 1.3.1 ou superior.

2.3.1.3 *RIKTEXT - Rule Induction Kit for Text*

O *RIKTEXT* é um pacote de softwares completo para categorização de documentos baseado em regras de decisão.

Seu objetivo é determinar o melhor conjunto de regras para a predição e classificação, onde o melhor conjunto é formado pelo menor número de regras com erro mínimo.

Os dados para este classificador devem estar na forma de tabela, onde cada linha corresponde a um documento e cada coluna corresponde a um termo do dicionário.

Cada célula da planilha recebe valores booleanos indicando a presença ou a ausência da palavra no documento, ou a freqüência linear do termo (número de vezes que o termo aparece no documento).

O *RIKTEXT* complementa o *TMSK*, disponibilizando métodos para construção e uso de regras para classificação de documentos. O formato dos dados de entrada do *RIKTEXT* é idêntico ao dos métodos de classificação apresentados no *TMSK* e as exigências mínimas para a execução do *RIKTEXT* são as mesmas do *TMSK*.

2.3.1.4 *Intext Software*

Intext é uma ferramenta de domínio público para análise de textos. Possui várias funcionalidades que permitem analisar o conteúdo de documentos em inglês e alemão.

Intext é considerado um *toolbox* com várias funcionalidades de análise do texto. Pode ser usado para listar *strings* que ocorrem no texto e suas freqüências; analisar conteúdos através da busca de padrões no texto, onde a saída é a lista de ocorrências do padrão identificado pelo usuário; listar referências cruzadas; permutar vocábulos.

Os requisitos mínimos de *hardware* e de *software* são: (1) Sistemas operacionais Win9x, WinME, Win2000, WinNT 4.x e Win XP; (2) Memória: 32 *Mbytes* disponível (no mínimo); (3) Espaço em disco de 5 *Mbytes* (no mínimo); (4) Textos em: inglês e alemão (espanhol em desenvolvimento); (5) Versões para MacOS, Linux e Unix (HP, IBM, Sun) encontram-se, atualmente, em desenvolvimento.

2.3.2 Ferramentas Comerciais

2.3.2.1 *TextSmart*

O *TextSmart* é um *software* que efetua análise qualitativa dos dados, realizando a mineração de textos sobre registros individuais que descrevem um problema do tipo perguntas e respostas. Normalmente, os registros correspondem ao texto referente a perguntas cujas respostas não têm limite de tamanho. Utiliza estatísticas para analisar automaticamente as palavras-chave e agrupá-las dentro de categorias afins. Também faz uso de uma lista de termos excluídos, que podem ser considerados como *stopwords*, assim como uma lista de *alias* ou termos sinônimos e um conjunto de regras para categorizar os textos. Tanto as listas de *stopwords* e *alias* quanto o conjunto de regras são fornecidos pelo usuário, e, a cada

término de processamento as listas podem ser revisadas, com o objetivo de se obter maior refinamento na resposta. Arquivos ".txt" são utilizados como entrada de dados.

O *TextSmart* disponibiliza os seguintes métodos de categorização de textos:

(i) Somente agrupamento: este método baseia seus resultados em *clusters* dos termos, que são determinados usando a forma de *clustering* hierárquico (*Furthest Neighbor Clustering*). Primeiro é calculada a similaridade, usando a medida *Jaccard* para cada par de termos. A medida *Jaccard* é baseada na freqüência relativa com a qual um par de termos ocorre ao mesmo tempo nas respostas. Para isso, é necessário contabilizar a presença ou ausência dos pares de termos, através das respostas e armazená-la numa tabela para todos os termos;

(ii) Somente freqüência dos termos: este método apenas cria categorias que contém um termo, começando com o termo que ocorre mais freqüentemente nas respostas, até o número máximo de categorias especificadas pelo usuário, ou até todos os termos serem categorizados. Esta freqüência indica apenas se o termo aparece na resposta, e não quantas vezes ele aparece;

(iii) Agrupamento e freqüência do termo: este método usa ambos os métodos descritos anteriormente para criar as categorias. Os termos que não fazem parte de nenhuma categoria são atribuídos individualmente a categorias de um único termo.

2.3.2.2 *STATISTICA Text Miner*

O *STATISTICA Text Miner* é uma extensão opcional do *STATISTICA Data Miner* com funcionalidades para seleção de recuperação de texto, pré-processamento e procedimentos analíticos/interpretativos de mineração de textos não-estruturados (incluindo páginas *Web*).

O aplicativo utiliza algoritmos para processamento analítico e interpretativo, a partir da conversão do texto não-estruturado, inclusive páginas *Web*, em uma matriz de termos.

A análise é feita a partir da redução de palavras ao seu próprio radical, da eliminação de termos sem significado (*stopwords*), além de parametrizações lingüísticas. Assim, o aplicativo elimina uma parte significativa do texto a ser analisado, reduzindo o número de termos da matriz de palavras de texto. Uma vez definidas as palavras com real valor para a análise, são aplicados algoritmos de mineração de textos, por meio dos quais derivam-se modelos

preditivos para explicar a possibilidade de ocorrência de termos significantes em outros documentos da mesma natureza.

As características principais do *STATISTICA Text Miner*:

(i) Contém várias opções de acesso ao texto em formatos diferentes, incluindo TXT, PDF, PostScript, HTML, XML, e na maioria dos formatos do *Microsoft Office* (por exemplo: DOC, RTF);

(ii) Provê suporte à "*Web-crawling*", permitindo extrair documentos de páginas *Web*;

(iii) Inclui *stoplists* e algoritmos de *stemming* para as línguas: dinamarquês, holandês, inglês, francês, alemão, italiano, português, espanhol, sueco etc. As *stoplists* podem ser editadas pelo usuário conforme necessário. O *software* é projetado de modo que o suporte à línguas adicionais possa ser feito com o mínimo de esforço;

(iv) Efetua a contagem de freqüência de todas as palavras nos documentos, que serve de base para todas as análises numéricas subseqüentes. Filtros adicionais podem ser aplicados. Por exemplo, cálculo de freqüência normalizada dos termos, freqüência inversa dos documentos, gerando uma sumarização numérica dos documentos;

(v) Disponibiliza métodos de análise estatística que são aplicadas sobre os sumários numéricos dos documentos, técnicas de *clustering*, tais como *kmeans* para identificar documentos similares e técnicas de predição que podem ser usadas para relacionar documentos a indicadores de interesse. Por exemplo, detecção de fraude, diagnósticos médicos, entre outros.

2.3.2.3 *SQL Server 2005*

O *SQL Server 2000* já disponibiliza algoritmos de classificação e de *clustering* de dados. Na nova versão *SQL Server 2005*, são implementados não só a execução desses algoritmos, como também componentes para análise de textos, podendo ser usados em conjunto com os algoritmos de mineração de dados [70].

O *SQL Server 2005* disponibiliza o *SQL Server Integration Services* (SSIS) que possui componentes para mineração de textos e profunda integração com os algoritmos de mineração de dados implementados. Os componentes para mineração de textos permitem identificar relacionamentos entre categorias de negócios e os dados (palavras e frases). Além

disso, a partir da descoberta de termos-chave no texto, encontra automaticamente termos de interesse.

Com os componentes do SSIS o desenvolvedor pode manipular documentos de texto e efetuar a tokenização, que divide os documentos em palavras e frases, inserindo-as no banco de dados. Outro componente existente calcula as freqüências dos termos e a freqüência inversa do documento. Com os valores das freqüências o SSIS cria os vetores compostos por pares (termos e pesos), que descrevem o conteúdo de um documento ou a categoria a qual ele pertence. A partir daí, nove algoritmos, incluindo árvores de decisão e de regressão, *clustering*, regressão logística e linear, redes neurais, *naïve bayes*, associação, *clustering* de seqüências e séries temporais estão disponíveis para a geração dos modelos e relatórios.

2.3.2.4 *LexiQuest Categorize*

É uma ferramenta que automatiza o processo de localização dos itens dos documentos ou parte deles em uma taxonomia, em uma ou mais categorias. Utiliza técnicas de processamento de linguagem natural (NLP), analisando o texto baseado na sua estrutura gramatical e o contexto. Com isso, apresenta grande eficiência no agrupamento de tipos de texto similares.

É capaz de processar textos em diversas línguas (inglês, francês, espanhol, italiano, alemão e holandês). Além disso, utilizando filtros embutidos, a ferramenta efetua a conversão de textos em HTML, PDF, ASCII e XML e formatos Microsoft (.doc, .xls, .ppt) para um formato único e próprio. Textos importados de bancos de dados e de fontes ODBC também podem ser convertidos. A identificação da linguagem escrita é feita usando *ngrams*, que são combinações de palavras ou caracteres de tamanho *n*.

Capítulo 3

Grids Computacionais e Portais

Segundo Foster et al. [71], *Grid* Computacional, também conhecido como grade ou grade computacional, é uma infra-estrutura que envolve o uso colaborativo e integrado de computadores, redes, bases de dados e instrumentos científicos pertencentes a diferentes organizações. O enfoque deste tipo de ambiente é a resolução de problemas para o compartilhamento de recursos entre organizações multi-institucionais. O *grid* computacional é uma tecnologia emergente que mudou a forma de abordagem de problemas computacionais complexos. Assim como a Internet revolucionou a forma do compartilhamento de informações, o *grid* computacional, similarmente, revolucionou o compartilhamento de poder computacional e de armazenamento [72].

Em uma outra definição, Dantas [73] afirma que um ambiente de *grid* computacional é a forma mais recente do processamento geograficamente distribuído que tem como característica a grande infra-estrutura das redes, empregada para compartilhamento de vários recursos heterogêneos, como servidores, aglomerados de computadores, programas, dados e serviços computacionais pertencentes a diferentes organizações que compõem a configuração deste tipo de ambiente.

Segundo Foster e Kelsselman [74], a palavra *grid* tem sua origem no termo *electrical power grid* que denota uma rede de energia elétrica, proporcionando a geração, transmissão e distribuição da mesma. Esta rede é a infra-estrutura que possibilita o uso desta energia – o recurso, neste caso – de forma transparente, generalizada e confiável.

Segundo Bertis [75], *grids* computacionais criam a "ilusão" de um computador virtual, em grande escala, facilmente gerenciável e com um grande poder computacional e uma vasta

quantidade de dispositivos sendo compartilhados. Desta maneira, Nash [76] afirma que *grids* abstraem as conexões entre aplicações, servidores, bases de dados, máquinas, dispositivos de armazenamento, entre outros, considerando tudo isso como um serviço virtualizado. De fato, um *grid* computacional é um ambiente onde um indivíduo se conecta para obter serviços computacionais que agregam recursos sob demanda, de maneira análoga à rede elétrica, que disponibiliza energia sob demanda e esconde do usuário detalhes, como a origem da mesma e a complexidade da malha de transmissão e distribuição. Logo, a infra-estrutura *grid*, segundo Malaika et al. [77], conecta diversos recursos de *hardware*, *software* e dados através de uma rede, fornecendo um acesso flexível e seguro para aplicações e dados. Esta idéia é ilustrada na figura 3.1.

Figura 3.1: Compartilhamento de recursos computacionais heterogêneos através de um *grid* computacional

A infra-estrutura de *grid* deve permitir um acesso robusto aos recursos, através de serviços padronizados, com interfaces e parâmetros muito bem definidos. Um dos grandes desafios é encontrar uma maneira de tornar a heterogeneidade dos recursos transparente para os usuários e administradores sem comprometer o alto desempenho. A padronização é um fator primordial para o desenvolvimento de aplicações neste tipo de tecnologia.

Os *grids* são compostos por entidades de processamento denominadas “nós”. Um nó de processamento pode ser um simples computador ou um *cluster*, que podem estar dispersos geograficamente. Cada nó componente do *grid* está apto a executar tarefas distintas, de modo que as aplicações possam tirar o maior proveito possível deste tipo de ambiente.

Dentre os fatores que alavancaram o desenvolvimento dos *grids* computacionais pode-se citar:

(i) Alta dispersão geográfica dos pesquisadores, gerando a necessidade de se criar um ambiente computacional para compartilhar recursos e resultados dinamicamente;

(ii) Fácil escalabilidade, para poder acomodar uma maior quantidade de dados e poder computacional;

(iii) O menor custo envolvido;

Uma das grandes vantagens da utilização de *grids* computacionais, é que os mesmos podem ser configurados com *hardware* de baixo custo, não necessitando da utilização de máquinas de grande porte, como supercomputadores.

Os fundamentos da tecnologia de *grids* são estabelecidos em dois artigos clássicos sobre o tema:

(i) *Anatomy of the Grid: Enabling Scalable Virtual Organizations* [71]. Nesse artigo, os autores definem *grid* computacional, propondo um conceito inicial sobre sua arquitetura;

(ii) *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration* [78]. Esse artigo descreve a arquitetura e a infra-estrutura de *grids*.

Segundo Foster [79], existem três características básicas para que um sistema computacional possa ser considerado um *grid*:

(i) Utilização de padrões abertos, interfaces e protocolos de propósito geral: os protocolos e padrões abertos são essenciais para que os sistemas em *grid* possam realizar funções básicas como autenticação, autorização, descobrimento de recursos e acesso a eles, sem perder a capacidade de escalar e interagir com diferentes plataformas de *hardware* e *software*;

(ii) Recursos coordenados que não se sujeitam a um controle centralizado: os sistemas em *grid* podem englobar recursos entre os mais variados tipos, desde o *desktop* de um usuário até um supercomputador. Pode haver um controle local, porém não existe um controle central para todo o *grid*;

(iii) Prover o mínimo em qualidade de serviços, como segurança, tempo de resposta e disponibilidade.

Skillcorn [80], define quatro tipos de ambientes de *grid*, como citado por Parra [81]:

(i) Ambientes de *grid* computacional: representam a extensão natural de grandes sistemas distribuídos e paralelos. Eles existem para proporcionar alto desempenho computacional. São formados por um conjunto de computadores disponíveis que são acessados por usuários através de uma única interface com o objetivo de submeterem aplicações que necessitam de mais de um computador para serem executadas;

(ii) *Grids* de acesso: o enfoque deste tipo de *grid* é a construção de um ambiente virtual em que usuários, pertencentes a diferentes organizações, possam interagir como se estivessem usando uma única plataforma de *hardware* dedicada. O desempenho neste tipo de ambiente não é tão importante quanto nos ambientes de *grid* computacional [71];

(iii) *Grids* de Dados: tem como característica permitir que enormes volumes de dados sejam armazenados em repositórios e movimentados com a mesma facilidade que os volumes de dados pequenos;

(iv) *Grids* de centros de dados: este tipo de ambiente possibilita acessar e realizar computação sobre enormes repositórios de dados distribuídos que por algum motivo não podem ser armazenados em um único local [82]. Ambientes deste tipo atendem a demandas oriundas de aplicações para *data mining* distribuído.

Os quatro tipos de ambientes de *grid* apresentados acima convergem para funcionalidades semelhantes, tais como: descoberta de recursos, planos de execução, autenticação e segurança e, heterogeneidade de servidores e de formatos de dados. No entanto, diferem em relação à ênfase atribuída a cada uma destas funções [81].

Contudo, para que haja uma distinção efetiva dos ambientes, deve-se ter em mente o que estes realmente podem disponibilizar como recursos e facilidades, diferenciando-se, de fato, uns dos outros. Por exemplo, quando Skillcorn [80] separou "ambiente de *grid*" de "*grid* de acesso", sugeriu que a diferença, em última análise, é somente a capacidade de processamento. Ora, uma vez que *grids* são ambientes que, em sua essência, compartilham recursos e estão disponíveis para a submissão de diversos tipos de aplicações, observa-se que há uma contradição com a divisão proposta, entendo-se que os dois ambientes convergem para a mesma arquitetura.

3.1 Benefícios

Grids computacionais podem ser entendidos como uma verdadeira evolução da computação, uma vez que é mais uma das respostas tecnológicas à demanda crescente no mercado por distribuir as cargas de trabalho de maneira mais eficiente pelos equipamentos disponíveis, simplificando o gerenciamento e diminuindo os custos. Os benefícios desta tecnologia, segundo Bertis [75], são:

(i) Exploração de recursos subutilizados e recursos adicionais: teoricamente, uma aplicação pode ser executada em qualquer máquina que faça parte de um *grid*, uma vez que esta possua acesso a determinados recursos solicitados pela aplicação. Pode-se escolher esta máquina utilizando diversos parâmetros, como por exemplo a que estiver com menor carga de processamento ou a que possuir disponível certo tipo de dispositivo ou determinados dados;

(ii) Capacidade de processamento paralelo: *grids* podem ser perfeitamente utilizados para o processamento de algoritmos paralelos. Como exemplo, pode-se ter um *cluster* representando um nó do *grid*, preparado para a execução de algoritmos paralelos.

(iii) Disponibilização de dispositivos e organizações virtuais: a colaboração entre os mais diversos tipos de usuários e aplicações é outra capacidade que pode ser desenvolvida com o advento dos *grids*. Recursos e máquinas podem ser agrupados e trabalharem juntos formando o que pode ser chamado de uma Organização Virtual (VO). Segundo Foster et al. [71], uma VO é uma entidade que compartilha recursos através do *grid*, utilizando uma determinada política de compartilhamento. Organizações Virtuais podem ser definidas conforme a área a qual atendem, variando em seus objetivos, escopo, duração, estrutura etc;

(iv) Confiabilidade: o próprio conceito de *grids* já atende a esta característica, uma vez que as máquinas podem estar geograficamente dispersas, de maneira que quando uma falhar, outras podem assumir o seu lugar. Deste modo, processadores, discos e outros dispositivos não são duplicados, portanto não são criadas redundâncias, como em um ambiente normal. Por isso, certamente, o *grid* como um todo dificilmente se tornará indisponível.

3.2 Desafios

Um grande problema encontrado atualmente em relação ao uso de *grids* é que, apesar de todas as pesquisas na área, ainda é extremamente difícil para os pesquisadores fazerem uso efetivo das ferramentas disponíveis. O fato é que as ferramentas são extremamente complexas. Muitas delas são monolíticas e demandam um grande esforço de instalação. Em [83] foram observados alguns problemas, discutidos a seguir.

Um *middleware de grid* deve fornecer vários serviços tais como autenticação, descoberta de recursos, gerenciamento de solicitações etc. No entanto, um sistema monolítico disponibiliza todas essas facilidades em uma única unidade, o que pode ser difícil para um usuário que apenas deseja fazer uso de uma pequena funcionalidade.

Um outro problema identificado é o das dependências. Bibliotecas *open-source* já são um padrão para o desenvolvimento destes sistemas. Porém, geralmente estes sistemas requerem modificações e versões não padronizadas das mesmas. Desta maneira, os administradores dos sistemas tem que manter múltiplas instalações diferentes de certas bibliotecas, acarretando, por vezes, sérios problemas de gerenciamento.

Depois que o *middleware* é instalado, o mesmo deve ser configurado. A configuração dessas ferramentas não é uma tarefa trivial, exigindo a edição de vários arquivos, emissão de certificados, mapeamento apropriado das máquinas etc, o que na verdade gera uma espécie de *overhead* burocrático. Ferramentas que facilitam tais procedimentos são necessárias.

3.3 Componentes

3.3.1 Recursos

Dentre os recursos disponibilizados por um *grid* computacional, pode-se destacar:

(i) Computação: o recurso mais disponibilizado em um *grid* computacional são seus ciclos de CPU. Em um *grid*, os processadores de seus nós variam em velocidade, tecnologia, plataforma de *software*, arquitetura etc. As aplicações que exploram esses recursos computacionais podem ser executadas de formas variadas;

(ii) Armazenamento: o espaço de armazenamento de cada nó integrante do *grid* incrementa a sua capacidade de armazenamento como um todo. *Grids* que possuem como

principal característica o compartilhamento do espaço de armazenamento são chamados de *Data Grids*;

(iii) Comunicações: conexões internas e externas podem ser compartilhadas através de um *grid*. Por exemplo, algumas aplicações necessitam processar grandes massas de dados e, estas podem não se encontrar no nó de execução, sendo necessária a sua busca em outra máquina. Assim, os dados devem ser transmitidos via rede, o que pode influenciar no desempenho da aplicação. Para evitar gargalos, máquinas com conexões ociosas podem ser utilizadas para enviar partes destes dados.

3.3.2 Arquitetura

A arquitetura de *grids* computacionais define os componentes fundamentais desta tecnologia e especifica a função e interação entre os componentes. Essa arquitetura é apresentada em uma estrutura em camadas, conforme ilustrado na figura 3.2 e descritas a seguir.

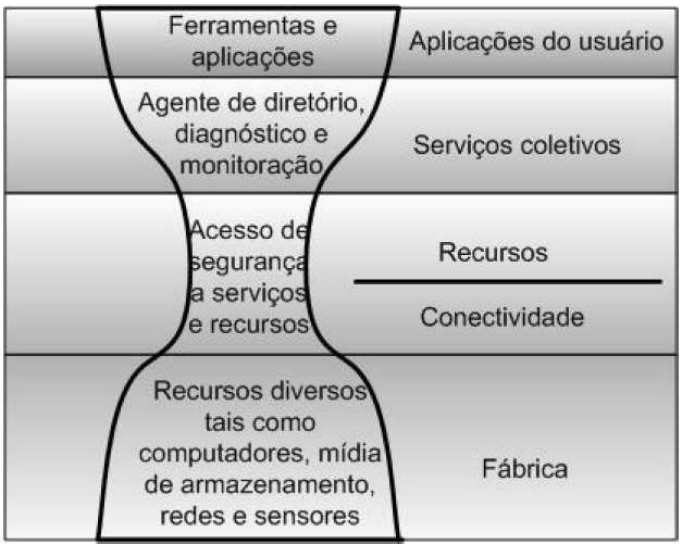


Figura 3.2: Arquitetura de um *grid* computacional. Fonte: Foster e Kelsselman [74]

3.3.2.1 Camada de Fábrica

A camada de fábrica engloba os recursos para os quais os acessos compartilhados são mediados pelos protocolos do *grid*. Exemplos dos elementos dessa camada são os diversos recursos, que podem ser lógicos ou físicos, tais como sistemas de armazenamento, discos virtuais, recursos de rede, sensores, computadores ou *clusters* de computadores. Os compo-

nentes da camada de fábrica executam as operações locais que são específicas dos recursos como resultado das solicitações de compartilhamento das operações de nível mais alto.

Os desafios nessa camada estão relacionados com a implementação de mecanismos internos nesses recursos que permitam, por um lado, a descoberta de sua estrutura, estado e capacidade e, por outro lado, o controle da qualidade de serviço, como mostram os exemplos que se seguem [72]:

(i) Recursos computacionais: nos recursos computacionais é necessária a definição de mecanismos para iniciar, monitorar e controlar a execução de processos. Funções internas são necessárias para determinar as características de *hardware* e *software* assim como as informações de estado relevantes, tais como carga de trabalho corrente;

(ii) Recursos de armazenamento: nos recursos de armazenamento é necessário o desenvolvimento de novos mecanismos para gerenciamento e acesso de arquivos. Geralmente são mecanismos para leitura, escrita e execução de arquivos remotos;

(iii) Recursos de rede: nos recursos de rede é importante o desenvolvimento de mecanismos de gerenciamento que forneçam controle sobre os recursos alocados para as transferências na rede. Funções internas devem ser providas para determinar as características e a carga na rede.

3.3.2.2 Camada de Conectividade

A camada de conectividade define os protocolos de comunicação e de segurança necessários para transações de rede específicas de *grid*, permitindo troca de informações entre recursos da camada de fábrica. Fornece também protocolos de autenticação e serviços de segurança criptografada para verificar a identidade de usuários e recursos. As necessidades supridas por esta camada incluem transporte, roteamento e nomeação. Estes protocolos são mapeados para uma pilha de protocolos TCP/IP. Desafios futuros devem abordar o projeto de novos protocolos de comunicação específicos para *grids* que atendam às demandas da dinâmica de redes específicas como, por exemplo, redes ópticas ou redes sem fio.

Os protocolos de segurança também estão sendo baseados em padrões, dentro do contexto da suíte de protocolos Internet. Os temas de segurança para *grids* computacionais que são mais importantes atualmente referem-se a:

(i) Autenticação única: um usuário deve se autenticar somente uma vez, dispensando autenticações sucessivas para acessos a recursos ou domínios administrativos diferentes;

(ii) Delegação: um usuário deve ter o poder de delegar a execução de um programa para os recursos onde ele tem autorização de uso. O programa deve ser capaz de delegar seus direitos para outro programa;

(iii) Integração com soluções de segurança local: as soluções de grids computacionais devem interoperar com soluções de segurança local;

(iv) Relacionamento de confiança baseada no usuário: caso o usuário tenha permissão para executar programas nos recursos A e B, não deve ser obrigatório que A e B interajam, para verificar permissões mútuas.

3.3.2.3 Camada de Recursos

O objetivo da camada de recursos é definir protocolos para negociação, inicialização, gerenciamento, controle e contabilização de operações de compartilhamento em recursos individuais de forma segura. A camada de recursos é construída sobre os protocolos da camada de conectividade. As implementações dos protocolos desta camada são baseadas nas funções da camada de fábrica para acessar e controlar os recursos locais. Os protocolos da camada de recursos mantém seu foco nos recursos individuais e ignoram o estado global. Duas classes principais de protocolos da camada de recursos são descritas:

(i) Protocolos de informação: são usados para obter informação sobre a estrutura e o estado de um recurso, como por exemplo, sua configuração, carga de trabalho corrente e política de uso e custo;

(ii) Protocolos de gerenciamento: são usados para negociar acesso a recursos compartilhados, especificando, por exemplo, os requisitos do recurso e as operações a serem executadas, tais como criação de processo e acesso a dados. Deve-se observar que ao utilizar este tipo de protocolo, é necessário saber se as operações solicitadas estão de acordo com a política do recurso a ser compartilhado.

A arquitetura do *grid* tem a forma de uma ampulheta, pois no gargalo da ampulheta deve haver um número reduzido de protocolos. Um conjunto de comportamentos de alto

nível (parte acima do gargalo) pode ser mapeado para muitas e diferentes tecnologias (parte abaixo do gargalo) [72].

3.3.2.4 Camada de Serviços Coletivos

Diferentemente da Camada de Recursos, que interage com um recurso simples, esta camada fornece protocolos e serviços que não estão associados a um recurso específico e sim a coleções destes. Pelo fato dos componentes coletivos serem construídos acima da camada de recursos e, portanto, participarem de uma camada mais "larga" do modelo, na camada de serviços coletivos pode ser implementada uma grande variedade de serviços, sem adicionar novos requisitos aos recursos que estão sendo compartilhados. Para isto, ela disponibiliza algumas facilidades de compartilhamento:

(i) Serviços de diretório: facilita aos integrantes de Organizações Virtuais descobrirem a existência e propriedades de recursos compartilhados. Estes recursos podem ser localizados por nome e outros atributos como tipo, disponibilidade e carga;

(ii) Serviços de co-alocação, escalonamento e *brokering*: permitem aos participantes de uma VO requisitar a alocação de um ou mais recursos para uma proposta específica e escalonar as tarefas nos recursos apropriados;

(iii) Serviços de co-alocação e agendamento: permitem que recursos possam ser alocados para determinado propósito e o agendamento de tarefas em recursos apropriados;

(iv) Serviços de monitoração e diagnóstico: monitoram recursos a fim de detectar falha, intrusão e sobrecarga;

(v) Serviços de replicação de dados: suporte a gerenciamento dos recursos de armazenamento para maximizar a performance de acesso a dados. Pode-se monitorar tempo de resposta, confiabilidade e custos;

(vi) Sistemas de programação compatíveis com *grids*: habilita a utilização de modelos de programação conhecidos no ambiente de *grids*, usando serviços como busca e alocação de recursos, segurança etc. Um exemplo seria a implementação de *Message Passing Interface* (MPI) no ambiente de *grid*.

3.3.2.5 Camada de Aplicações

Nesta camada estão as aplicações do usuário que operam dentro de um ambiente de uma VO. As aplicações são construídas nos termos dos serviços oferecidos em cada camada da arquitetura. As ferramentas no nível da camada de aplicação devem oferecer mecanismos para que os usuários consigam escrever e executar suas aplicações no ambiente de *grid*.

3.4 Serviços Web

Serviços Web ou *Web Services* é uma tecnologia de computação distribuída que permite a criação de aplicações cliente/servidor usando como plataforma de comunicação protocolos abertos e amplamente conhecidos como o HTTP e o XML. Desta maneira, pode-se executar um serviço através de uma Intranet ou mesmo da Internet, ocultando do cliente detalhes relativos a implementação (serviço desenvolvido em Java, C++ etc) e plataforma (servidor Linux, Windows, Unix ou em um *mainframe*), bastando ao cliente conhecer a URL através da qual o serviço pode ser acessado e as interfaces de seus métodos.

Os serviços *web* são definidos pelo W3C (*World Wide Web Consortium*), definindo uma maneira padronizada para descrevê-los – o WSDL (*Web Service Description Language* – Linguagem de Descrição para Serviço Web). Basicamente este padrão define como são descritos os métodos, parâmetros, tipos de dados, protocolo de transporte e URI (*Uniform Resource Identifier* – Identificador Uniforme de Recurso) de um serviço. Para um serviço *web* ser acessado, o mesmo precisa ser publicado em um servidor. Também deve existir uma maneira pela qual possam ser realizadas pesquisas em busca de um determinado serviço. Para suprir estas necessidades, o W3C definiu um padrão chamado UDDI (*Universal Description, Discover and Integration* – Descrição, Descoberta e Integração Universal). Um protocolo padrão para comunicação também foi definido, o SOAP (*Simple Object Access Protocol* – Protocolo Simples de Acesso a Objeto), sendo baseado em XML e usando o HTTP como protocolo de transporte.

Serviços *web* também possuem desvantagens. Uma delas é o *overhead*, pois a transmissão de dados é feita usando XML que é, obviamente, menos eficiente do que usar um código binário proprietário. Tal como Java, o que se ganha em portabilidade perde-se em eficiência. Porém, este *overhead* não configura-se como um problema para a maioria das aplicações,

mas segundo Sotomayor [84], é difícil encontrar aplicações críticas de tempo real que usem serviços *web*.

Entretanto, existe uma importante característica que distingue serviços *web* das outras tecnologias de computação distribuída. Enquanto tecnologias tal como CORBA e EJB são voltadas para sistemas distribuídos altamente acoplados, onde cliente e servidor são muito dependentes um do outro, serviços *web* são mais adequados para sistemas fracamente acoplados, onde o cliente pode não ter conhecimento, *a priori*, do serviço até que ele o invoque. Sistemas distribuídos altamente acoplados são ideais para aplicações Intranet, e não para aplicações *web*. Segundo Sotomayor [84], serviços *web* são mais adequados para alcançar as demandas de uma aplicação *Internet-wide*, tal como aplicações orientadas a *grid*.

Nesse paradigma, as funcionalidades oferecidas pelas aplicações de negócios são encapsuladas dentro de serviços *web*: componentes de *software* descritos em um nível semântico, que podem ser chamados por uma aplicação ou por outros serviços através de uma camada Internet padrão como http, XML, SOAP, WSDL, e UDDI. Uma vez disponibilizado, um serviço *web* possibilita que várias organizações possam estar interconectadas para implementar atividades colaborativas.

Em outras palavras, a tecnologia de serviço *web* pode ser tratada como um modelo que permite integração entre diversos aplicativos e serviços na Internet. Na visão da engenharia de *software*, um serviço *web* nada mais é do que um repositório remoto de componentes.

3.4.1 Acesso a Serviços

Os serviços *web* surgiram como uma nova tecnologia para computação distribuída, desenvolvida tirando proveito de vários padrões já bem estabelecidos, como RMI e CORBA. A principal característica de um serviço *web*, é a camada de transporte, que utiliza o protocolo HTTP para envio de mensagens entre o cliente e o provedor. Na figura 3.3 é ilustrado como ocorre essa comunicação, utilizando serviços *web*.

(1) O cliente utiliza o Registro UDDI para descobrir onde o serviço desejado está sendo fornecido;

(2) O Registro UDDI responde ao cliente com o endereço do serviço na forma de um URL que aponta para o servidor que fornece o serviço desejado. Um exemplo de URL: http://localhost:8080/Aiuri2;

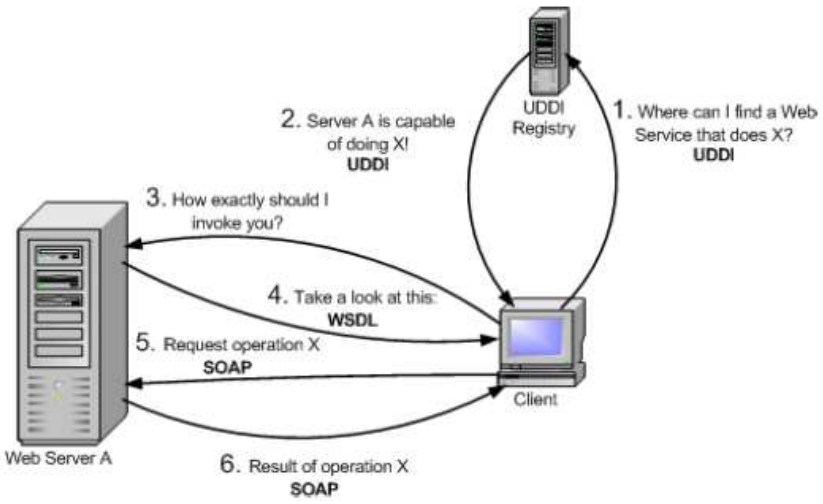


Figura 3.3: Comunicação entre cliente e provedor usando serviços *web*

(3) Com esta informação, o cliente conhece a localização do serviço, mas ainda não sabe como invocá-lo. Então, o cliente pergunta ao servidor como pode invocar o serviço desejado;

(4) O servidor responde ao cliente enviando um documento WSDL, que descreve a interface do serviço, ou seja, seus métodos, parâmetros e tipos de dados;

(5) De posse destas informações, o cliente sabe como invocar o serviço, sendo que esta invocação pode ser realizada através de diversos protocolos. O protocolo padrão para esta função é o SOAP. O cliente então faz a chamada ao serviço usando este protocolo;

(6) O serviço responde então com uma resposta SOAP, encapsulando os dados no padrão XML.

Como mencionado acima, serviços *web* é a tecnologia escolhida para aplicações distribuídas cujos clientes e servidores são fracamente acoplados. Isto faz com que seja a escolha natural para a construção de aplicações baseadas em *grid*. Entretanto, é importante ressaltar que os serviços *web* possuem limitações, que podem causar alguns problemas em aplicações voltadas para *grids*. Para contornar essas limitações, surgiram os serviços *grid*.

3.5 Serviços Grid

Serviços Grid ou *Grid Services* é uma tecnologia construída com base nos conceitos e tecnologias de *Grid* e serviços *web*. É uma arquitetura que:

(i) Define semânticas uniformes de exposição (serviços *grid*);

(ii) Define mecanismos padrões para criar, nomear e descobrir instâncias transientes de serviços em *grid*;

(iii) Fornece transparência de locação e múltiplos protocolos (*bindings*) para instâncias de protocolos;

(iv) Fornece uma integração entre as camadas mais baixas das plataformas.

O *Open Grid Service Architecture* (Arquitetura Aberta de Serviço Grid) e *Open Grid Service Interface* (Interface Aberta de Serviço Grid) (OGSA/OGSI), detalhados nas seções 3.6.2 e 3.6.1, definem um serviço *grid* como um serviço *web* que fornece um conjunto de interfaces que seguem convenções específicas. As interfaces são responsáveis pela descoberta, criação de serviços dinâmicos, gerenciamento e notificação.

As convenções e interfaces que definem um serviço *grid* referem-se ao comportamento relacionado ao gerenciamento de instâncias de serviços dinâmicos. Cada participante de uma organização virtual, ao invés de adotar um conjunto estático de serviços persistentes, contendo complexas atividades de requisição aos clientes, pode instanciar os serviços de transação de maneira dinâmica. As transações dinâmicas envolvem o gerenciamento e interações associadas com as particularidades do estado de tarefas requisitadas, quando o estado da tarefa termina, o serviço pode ser destruído.

Portanto, é uma tecnologia que teve como origem uma demanda para integrar serviços através de organizações virtuais, heterogêneas e dinâmicas, formadas por recursos distintos, seja dentro de uma mesma empresa seja por recursos externos compartilhados. Uma integração que pode ser tecnicamente desafiadora devido à necessidade de se atingir diversos tipos de qualidade de serviços quando se executa em plataformas diferentes.

3.5.1 Descoberta de Serviços

Mesmo sendo possível que clientes acessem serviços diretamente, sem a utilização de um catálogo, é de fundamental importância construir uma infra-estrutura de serviços sob demanda que permita que os serviços sejam descobertos dinamicamente.

Desta maneira, em um ambiente de *grid* computacional, é fundamental que seja possível descobrir os serviços existentes que possam atender a uma determinada aplicação. A idéia é

que um serviço de descoberta funcione como as páginas amarelas de um catálogo telefônico, que permitem aos usuários da rede telefônica encontrarem prestadores de serviços, a partir de alguns critérios, como classificação da atividade, localização do estabelecimento e outras informações divulgadas no catálogo.

3.5.2 Autenticação e Autorização

Fazendo uma comparação de *grids* computacionais com outras plataformas, fica evidente que a ampla dispersão de serviços e clientes gera a necessidade de um maior controle sobre o acesso aos recursos e serviços. Por exemplo, em uma rede local, ao efetuar *login* no sistema, o usuário é identificado e autenticado, em geral, para todos os recursos conectados e serviços disponíveis na rede, onde normalmente se mantém um cadastro de usuários que é válido para toda a rede [85].

Porém, em um ambiente variado e disperso, como um *grid* computacional, é necessária uma forma de acesso a cada recurso, ou conjunto de recursos, de maneira a não interferir nas políticas de segurança dos sítios locais. As formas de acesso devem oferecer garantias sobre autenticação dos usuários, uma vez que, do contrário, os administradores dos sítios não terão a garantia de que usuários realmente "idôneos" estarão acessando recursos de seus domínios.

A solução para isso foi a implementação dos certificados digitais, que são baseados nos conceitos de chave pública e privada. Deste modo, cada domínio administrativo pode manter sua política local de autenticação e autorização, e prover um serviço que cuida da autenticação e autorização de clientes externos.

3.5.3 Privacidade de Dados

A garantia da privacidade dos dados é um aspecto desafiador, na medida em que *grids* computacionais são ambientes altamente dispersos e dinâmicos. A demanda por privacidade dos dados é fundamental para alguns clientes.

Assim, ainda existem muitos problemas em aberto nessa área. Hoje, instituições que necessitam de um serviço de execução distribuído e têm como requisito principal a privacidade, tem restrições quanto à utilização de uma infra-estrutura tão dispersa quanto o *grid*. Essas aplicações executam em ambientes controlados e proprietários onde a privacidade pode ser garantida. Um exemplo disso é a infra-estrutura montada pela HP para prover a renderização

das imagens do filme *Sherek 2* [86].

3.5.4 Composição de Serviço

Em *grids* computacionais os serviços podem ser compostos por outros serviços. Pode-se imaginar um serviço composto como um pacote de viagem adquirido junto a uma empresa, em que, para o cliente, a empresa junto a qual ele fez a aquisição é a responsável direta pela implementação dos serviços que constam no pacote. Porém, é fato, que a maioria dos serviços são prestados por outras empresas, mas para o usuário final, isso é indiferente.

Neste contexto, a composição de serviços traz uma série de benefícios para a computação distribuída, dentre eles: (i) abstração da complexidade do serviço para o cliente; (ii) reutilização das funcionalidades implementadas por outros serviços.

De fato, quanto mais complexo for o serviço, menos envolvido o cliente desejará estar. Assim, quanto mais compostos forem os serviços, mais simples e de alto nível devem ser as aplicações a serem disponibilizadas para os clientes.

Um outro aspecto em relação à composição de serviços é a reutilização de código, de maneira que um serviço já desenvolvido e disponível no ambiente, possa ser usado na composição de novos serviços. Um exemplo desta implementação são os serviços *web* fornecidos pela Receita Federal que permitem a consulta e validação de CPF e CNPJ, permitindo que outras aplicações possam usá-los evitando que sejam implementados novamente.

Porém, este tipo de abordagem trás alguns desafios. A garantia da qualidade do serviço é um deles, visto que em sua composição existem serviços que estão fora do domínio de quem implementou o serviço. Para o usuário final, o serviço é um só, mas para o provedor este serviço é composto também por serviços prestados por terceiros, e este por sua vez é o responsável por tudo.

3.6 *Open Grid Forum* (OGF)

O *Open Grid Forum* (Forum Aberto de *Grid*) foi criado em setembro de 2006, resultante da fusão do *Global Grid Forum* (Forum Global de *Grid*) (GGF) e do *Enterprise Grid Alliance* (Aliança Empreendedora de *Grid*) (EGA). O GGF teve o seu primeiro encontro realizado em junho de 1999. O EGA foi Criado em 2004 com o foco exclusivo na aceleração da adoção

de *grid* em centro de dados.

Atualmente, é evidente a necessidade de evolução dos padrões de *grid*, com o intuito de popularizar o seu uso. Um aspecto importante a ser levado em consideração, é que além dos benefícios que a computação em *grid* está trazendo para a comunidade acadêmica, com o compartilhamento de recursos e serviços, outro fator que também colabora para a popularização de seu uso é a demanda da indústria por integração de sistemas comerciais. Essa demanda já impulsiona a convergência de diversas tecnologias, gerando uma grande necessidade de padronização. Devido a essa forte necessidade de padronização, é um consenso que o foco deve ser o desenvolvimento de tecnologias para *grids* computacionais que sejam capazes de integrar recursos e serviços distribuídos de maneira transparente e eficiente [85]. Assim, os *grids* evoluíram para utilizar uma abordagem orientada a serviços baseada em padrões de tecnologias para serviços *web*, ou seja, os já conhecidos serviços *web*.

Pode-se definir um *serviço computacional* como qualquer recurso (ou outro serviço) que possa ser acessado remotamente e descrito através de uma interface (por um provedor), que pode ser interpretada de forma automática por um cliente [85]. Desta maneira, qualquer recurso pode ser considerado um serviço, desde que exista a possibilidade de se criar uma abstração que forneça essa interface.

Um dos esforços para a padronização de *grids* é o GGF, que é um fórum de pesquisadores que utilizam e desenvolvem tecnologias e aplicações para *grid*. O GGF contém grupos de trabalho de várias áreas, onde cada uma delas trata de um problema particular. Algumas das áreas são serviços de informação, segurança, compartilhamento e gerenciamento, desempenho, arquitetura, dados e modelos e aplicações. Segundo Foster et al. [78], uma especificação GGF é a Arquitetura de Serviços *Open Grid*. O centro da arquitetura OGSA é o serviço *grid*. A OGSA gerou a OGSI que basicamente descreve as funcionalidades que serviços *grid* devem ter e que não são encontradas em serviços *web*.

3.6.1 OGSI

O OGSI (*Open Grid Services Infrastructure* – Infra-estrutura Aberta de Serviços *Grid*) [87], criado em agosto de 2004, tem o objetivo de definir as interfaces básicas e os comportamentos de um serviço *grid*, provendo mecanismos de localização de recursos, com seu gerenciamento e utilização em larga escala. OGSI é a materialização da arquitetura definida

pelo padrão OGSA, uma vez que os serviços especificados servem como base para construção dos *grids*. Mais detalhes podem ser encontrados em [88].

3.6.2 OGSA

OGSA (*Open Grid Services Architecture* – Arquitetura Aberta de Serviços Grid) foi introduzida por Foster et al. [78], sendo uma arquitetura de serviços básicos para a construção de uma infra-estrutura de *grid* computacional, tendo a sua primeira versão (v1.0) disponibilizada em janeiro de 2005. Um *grid* é composto por vários subsistemas, como: (i) descoberta e monitoramento de recursos; (ii) gerenciamento de VO; (iii) segurança; (iv) submissão e gerenciamento de tarefas, gerenciamento de dados, *workflow*, entre outros. Esta arquitetura tem como foco padronizar os diversos serviços que são encontrados em uma aplicação de *grid*, através da especificação de um conjunto de padrões e interfaces para estes serviços. Mais detalhes ver [89].

OGSA foi desenvolvido pelo GGF com o intuito de definir uma arquitetura padrão e aberta para aplicações baseadas em *grid*. A meta do OGSA é padronizar praticamente todos os serviços que são encontrados em uma aplicação de *grid*, através da especificação de um conjunto de interfaces para estes serviços.

No entanto, o grupo OGSA percebeu a necessidade de um *middleware* distribuído para conseguir criar essa nova arquitetura sobre a qual se apoiaria. Apesar de existirem vários *middlewares* distribuídos (por exemplo CORBA, RMI, RPC etc) o grupo optou pela arquitetura serviço *web* por esta apresentar melhores opções em relação às demais, sendo as características de fraco acoplamento e interoperabilidade os fatores mais importantes para *grids* computacionais. Por definição, sabe-se que o acoplamento é uma medida de força das interconexões entre componentes de um sistema, onde um baixo acoplamento implica que mudanças em um componente dificilmente afetarão outros componentes, ou terão menos impacto. Por outro lado, um baixo acoplamento indica um aumento na coesão, que também, por definição, é a medida da proximidade das partes de um sub-componente. Em um componente deve ser implementada uma única entidade lógica ou função. Isto implica no impacto que modificações em algum componente pode gerar. Ou seja, sistemas altamente coesos implicam em modificações localizadas com nenhum ou muito pouco impacto sobre o sistema.

Contudo, a arquitetura de serviços *web* ainda tem diversos incovenientes que a torna inadequada para as necessidades do OGSA [84]. Estes obstáculos foram superados através do OGSA, com a definição de um tipo estendido do serviço *web* chamado serviço *grid*. Um serviço *grid* é simplesmente um serviço *web* com extensões que o tornam adequado para a construção de aplicações de *grid*, conforme as necessidades do OGSA.

Durante a criação desta nova arquitetura, seus autores perceberam que seria necessário escolher como base algum tipo de *middleware* distribuído [90], uma vez que cada serviço deve ser acessado da mesma maneira, sem levar em consideração o fornecedor, organização, implementação interna etc. Esta base para a arquitetura poderia ser qualquer *middleware* distribuído, como CORBA, RMI etc. Porém, a opção escolhida foi serviço *web*, também, por ser a mais adequada para sistemas fracamente acoplados.

A arquitetura de serviços *web* faz uma separação clara entre a interface e a implementação. A interface é definida em uma linguagem chamada WSDL (*Web Services Description Language* – Linguagem de Descrição de Serviços *Web*) [91], que é uma linguagem independente de plataforma, especificada em XML. Portanto, serviços *web* podem lidar facilmente com sistemas heterogêneos e fazem com que OGSA tenha as facilidades de transparência desejadas.

As interfaces padronizadas definidas pelo OGSA facilitam a virtualização de serviços e recursos. Isso possibilita o uso de vários tipos de recursos de maneira transparente. Outra vantagem da virtualização é que as interfaces padronizadas permitem um baixo acoplamento entre o cliente e o provedor do serviço. Esse baixo acoplamento facilita a mudança na implementação dos serviços sem causar, necessariamente, mudanças na implementação do cliente, bem como o inverso.

3.6.2.1 WSRF

Uma vez que toda a base de serviços *grid* surgiu das tecnologias para serviços *web*, alguns aspectos da especificação OGSI precisavam ser refinados devido à evolução dos padrões de serviços *web*. Pode-se dizer que OGSI poderia se tornar uma padronização de serviços *web*, no entanto a mesma possui algumas desvantagens que impediram esta convergência.

Para resolver os problemas de OGSI e iniciar um processo de convergência entre *grid* e serviços *web*, um novo padrão foi definido em maio de 2006 para substituir o OGSI: o

WSRF (*Web Service Resource Framework – Framework* de Recursos de Serviço *Web*) [88]. WSRF é um padrão completamente baseado em serviços *web*. Este padrão foi criado com o objetivo de solucionar algumas desvantagens de OGSI, como segue [88] [90]:

(i) A Especificação OGSI é longa e densa: não possui uma separação clara de funções que dão suporte à adoção incremental. O WSRF soluciona este problema particionando OGSI em cinco padronizações (Banks [92]) e mais duas especificações complementares: *WSNotification* e *WSAdressing*;

(ii) OGSI não funciona bem com ferramentas de serviço *web*: são usados em demasia esquemas XML e documentos orientados a operações WSDL. Estes esquemas não utilizam necessariamente construções padronizadas. WSRF usa um esquema XML padronizado, sendo mais familiar aos desenvolvedores e é amplamente suportado por ferramentas já existentes;

(iii) OGSI é muito voltada à orientação a objetos: na OGSI um recurso *stateful* é modelado como um serviço *web* que encapsula estados de recursos, com a identificação e ciclo de vida de um serviço e estado de recursos acoplados. Recursos *stateful* podem ser definidos como componentes com manutenção de estado entre diversas chamadas remotas. O uso desses componentes é útil em diversas situações em que se deseja manter uma conversação e manutenção de estado do objeto. Um exemplo clássico deste tipo de recurso é o componente carrinho de compras de *sites web*.

No entanto, serviços *web* puros não têm estados ou instâncias, como também, algumas de suas implementações não acomodam a criação e destruição de serviços dinâmicos. Na especificação WSRF foi levado em conta a maioria das objeções feitas pela comunidade de serviços *web*, tornando mais fácil para as ferramentas atuais de serviços *web* fornecer suporte a WSRF. No WSRF a arquitetura OGSI é organizada de modo a fazer uma distinção explícita entre os serviços e entidades *stateful* que atuam sobre este serviço.

Atualmente o WSRF é o que há de mais novo para o desenvolvimento de *middleware* para *grids* computacionais. Porém, Humphrey et al. [93] fazem uma avaliação deste padrão e observam que, embora o potencial do WSRF seja considerável, algumas restrições ainda não foram resolvidas.

3.7 Globus Toolkit

Dentre os diversos *middlewares* para construção de *grids*, o *Globus Toolkit* é considerado por vários pesquisadores, centros de pesquisa e empresas, como o pacote mais completo e um padrão de fato. O Globus é fruto do projeto Globus, tendo sido desenvolvido inicialmente pelos pesquisadores Foster e Kesselman [17], que propuseram um conjunto de serviços que fossem utilizados pelas aplicações sem a necessidade de adaptação a um modelo particular de programação.

O projeto Globus atualmente é uma iniciativa que envolve diversas instituições de pesquisa e conta com o apoio de grandes empresas, como a IBM e a Microsoft. As principais instituições de pesquisa envolvidas incluem o Laboratório Nacional de Argonne (ANL), Universidade de Chicago, Universidade do Sul da Califórnia (USC) e o Laboratório de Computação de Alto Desempenho da Universidade do Norte de Illinois, todos esses situados nos Estados Unidos, além da Universidade de Edimburgo. O sistema de Computação em *Grid* desenvolvido no contexto do projeto Globus é denominado Globus *Toolkit* e provê uma série de funcionalidades que permitem a implantação e desenvolvimento de aplicações de sistemas de computação em *grid*.

Segundo Foster e Kesselman [94], os serviços no Globus são baseados em um modelo conhecido como *Hourglass* (Ampulheta). Nesta analogia, os serviços locais encontram-se na parte inferior da ampulheta, enquanto que os serviços globais estão na parte superior (vide figura 3.2). Os serviços fornecidos pelo Globus ficam localizados no gargalo da ampulheta. Os componentes do conjunto de ferramentas Globus podem ser usados tanto independentemente quanto em conjunto no desenvolvimento de aplicações e de ferramentas de programação em *grid*. Para cada componente existe uma *API* em linguagem C ou Java definida para facilitar o uso pelos desenvolvedores. Na Figura 3.4 é ilustrada esquematicamente a estrutura do *Globus Toolkit*.

O *Globus Toolkit* sofreu profundas transformações na transição entre a versão 2 (GT2) e 3 (GT3). Tais transformações não foram feitas apenas com o intuito de evoluir a caixa de ferramentas (*toolkit*), mas incluíram uma mudança completa nos módulos, interfaces e protocolos.

A versão 2.X ainda é muito utilizada. Esta versão é caracterizada por um conjunto de serviços para a construção de sistemas e aplicações de *Grid*, sendo adotada como um padrão

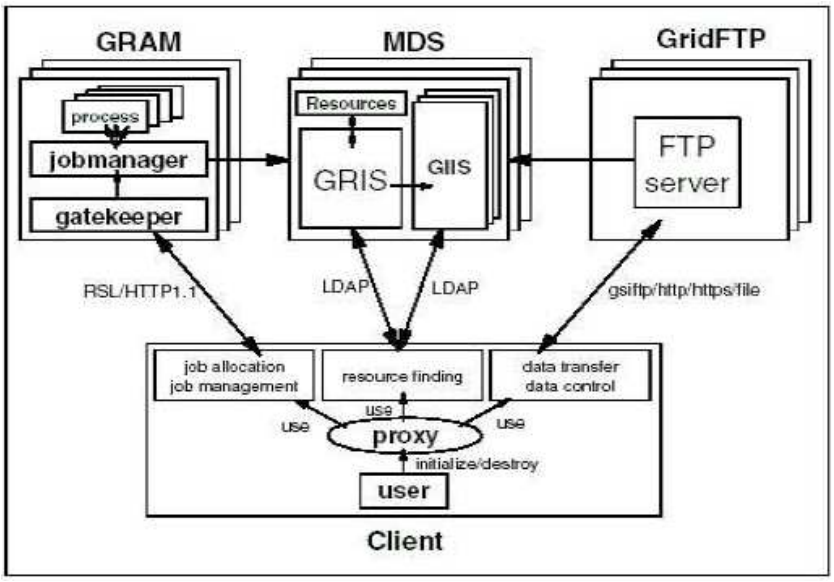


Figura 3.4: Visão geral do Sistema *Globus Toolkit*

de fato por muitos *middlewares*.

A partir da sua terceira versão, conhecida como *Globus Toolkit 3* (GT3), vem sendo desenvolvidas aplicações e infra-estruturas para computação distribuída, de acordo com a perspectiva da arquitetura de *software* SOA. Nesta versão, os serviços Globus são definidos como serviços *grid*, em conformidade com as especificações OGSA.

3.7.1 *Globus Toolkit 4*

Com a incorporação da especificação WSRF no *Globus Toolkit*, foi desenvolvida uma nova arquitetura totalmente baseada no OGSA, denominada *Globus Toolkit 4*, mais conhecida como GT4. Na figura 3.4 é ilustrada esta mudança arquitetural ocorrida da versão 3 (GT3) para o GT4.

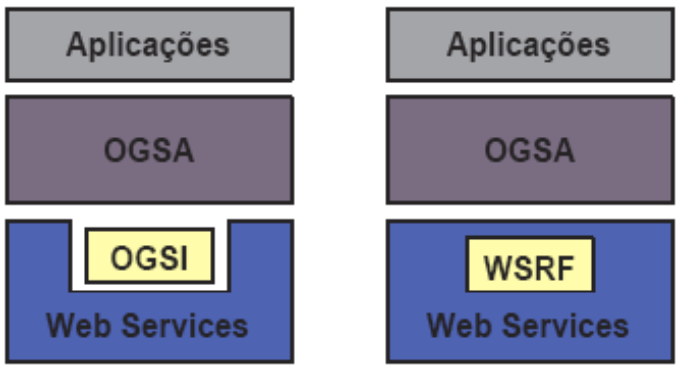


Figura 3.5: Mudança arquitetural: GT3 $\times$ GT4

Na figura 3.6 são ilustrados vários aspectos da arquitetura do GT4. Os três conjuntos de

componentes mais importantes são:

(i) Um conjunto de serviços (parte superior da figura) realiza toda a implementação de serviços, que podem estar relacionados ao gerenciamento de execução (GRAM), acesso e transferência de dados (GridFTP, RFT, OGSA-DAI), gerenciamento de réplicas (RLS, DRS), monitoração e descoberta de recursos (*Index*, *Trigger*, WebMDS), gerenciamento de credencial (*MyProxy*, Delegação, *SimpleCA*);

(ii) Três *containers* podem ser utilizados para a execução dos serviços em Java, Python e C, respectivamente. Estes containers tem a função de prover implementações de segurança, gerenciamento de descoberta de recursos, gerenciamento de estado e outros mecanismos frequentemente necessários quando se desenvolvem os serviços. Eles estendem as mais importantes especificações de código aberto para serviços *web*, incluindo WSRF, *WS-Notification* e *WS-Security*;

(iii) Um conjunto de bibliotecas possibilitam aos programas desenvolvidos pelos clientes em Java, C e Python, invocarem serviços do GT4 ou serviços implementados pelo próprio desenvolvedor.

É importante ressaltar que o GT4 é mais que apenas um conjunto de serviços úteis. O uso de abstrações e mecanismos uniformes significa que os clientes podem interagir com diferentes serviços de formas similares, o que facilita o desenvolvimento de sistemas complexos e interoperáveis, e encoraja o reuso de código.

3.8 *Grid EELA*

O projeto EELA (*E-Infraestructure Shared Between Europe and Latin America* – Infra-estrutura Compartilhada entre a Europa e América Latina) tem como objetivo estabelecer uma rede de colaboração humana para compartilhar uma infra-estrutura que suporta o desenvolvimento e teste de aplicações avançadas. Neste esforço coletivo, o EELA diponibilizou uma infra-estrutura comum na América Latina e na Europa, interconectada por meio das redes *Clara* e *Géant*, onde algumas aplicações de interesse geral estão implementadas: Biomedicina, Física de Altas Energias, e-Educação e Clima.

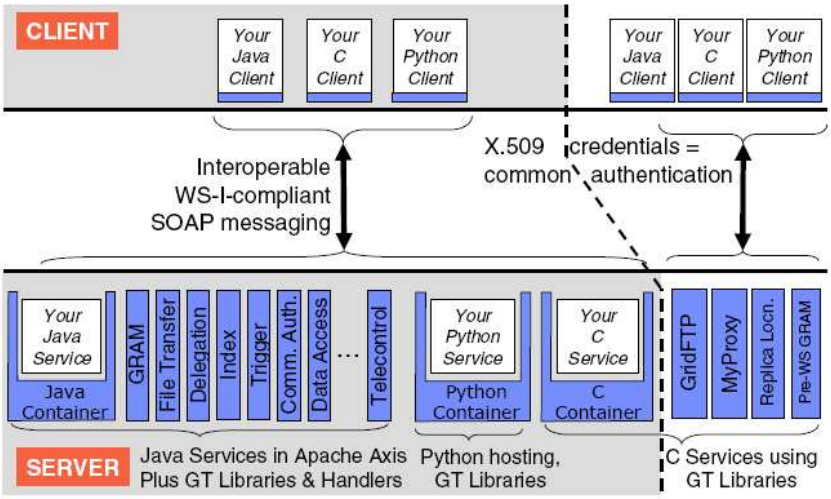


Figura 3.6: Esquema da arquitetura do GT4, mostrando os principais componentes. As caixas compartilhadas denotam código GT4; as caixas brancas representam código do usuário

Iniciada em janeiro de 2006, a infra-estrutura do EELA segue o modelo do projeto europeu EGEE (*Enabling Grids for E-science* – Capacitando *Grids* para E-Science) para o desenvolvimento e implantação de *grids* para uso científico, que atualmente forma uma grande "e-infra-estrutura" em operação no planeta. Atualmente o EELA compartilha mais de 1100 CPUs e 60 *Terabyes* de recursos de armazenamento, sendo baseado no LCG/gLite. Na figura 3.7 é ilustrada uma visão geral da área de atuação do projeto EELA e dos projetos co-participantes.

Figura 3.7: Visão global do *grid* EELA

3.9 Portais de *Grid*

No início da década de 2000, o que hoje é chamado de portal era conhecido como máquina de busca, cujo objetivo era facilitar o acesso às informações contidas em documentos espalhados pela Internet. Inicialmente, as máquinas de busca possibilitavam ao usuário da Internet localizar documentos a partir de pesquisas booleanas e navegação associativa entre *links*. Para reduzir ainda mais o tempo de busca na Internet e auxiliar os usuários menos experientes, vários *sites* de busca incluíram categorias, isto é, passaram a filtrar *sites* e documentos em grupos pré-configurados de acordo com seu conteúdo – esportes, meteorologia, turismo, finanças, notícias, cultura etc. O passo seguinte foi a integração de outras funções, como, por exemplo, as comunidades virtuais e suas listas de discussão, chats em tempo real, possibilidade de personalização dos *sites* de busca (My Yahoo!, My Excite etc.) e acesso a conteúdos especializados e comerciais. Essa nova concepção de máquina de busca é que passou a ser chamada de portal. Contudo, Reynolds e Koulopoulos [95] identificam as seguintes fases do progresso do portal *web*: pesquisa booleana, navegação por categorias, personalização e, por fim, funções expandidas para outras áreas dos mundos informacionais e comerciais.

Neste contexto, é importante ressaltar, que toda a tecnologia e teoria por trás dos *grids* deve possuir uma maneira amigável de utilização, tornando transparente para o usuário o seu uso. Em consonância com esta abordagem, surgem os portais de *grids*.

Por definição, um portal é um ambiente *web* seguro que possibilita que uma organização agregue e compartilhe conteúdo – informação, serviços e aplicações – com clientes, parceiros de negócios, empregados e fornecedores [96].

Atualmente, os portais podem ser usados para acessar um único servidor, um *cluster*, ou até mesmo um *grid* com uma infinidade de máquinas, criando oportunidades para mais organizações e seus usuários. Na figura 3.8 é ilustrada esta perspectiva.

Dentre as vantagens que podem ser atribuídas a um portal para *grids* é que todas as operações são transparentes para o usuário. Uma simples submissão de um *job* pode ser feita com o uso de um *browser*, bastando ao usuário possuir um certificado, que servirá como mecanismo de validação. Toda a complexidade envolvida na submissão permanece encapsulada. Além disso, as tarefas de administração são facilitadas, podendo ser feitas remotamente.

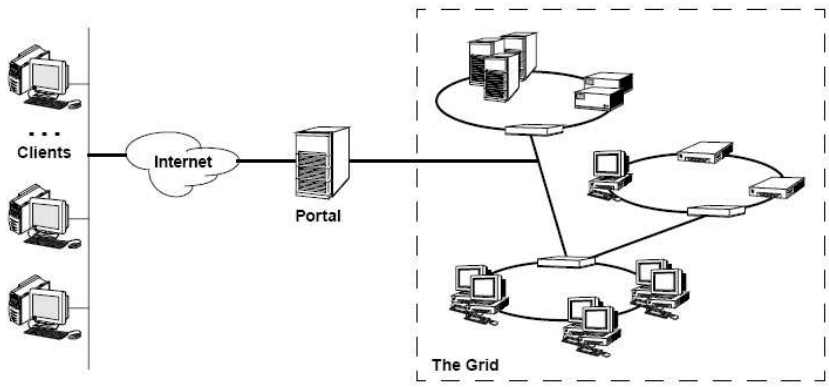


Figura 3.8: Acesso a um *grid* computacional através de um portal para *grid*

Como já mencionado, um mecanismo de validação é necessário para que os usuários tenham acesso aos recursos dos portais, de maneira que as atividades de autenticação e autorização fiquem encapsuladas e transparentes. Além de restringir o acesso a informações e tarefas, mecanismos de contabilização também são necessários com o objetivo de administrar o uso e a performance do ambiente como um todo.

Atualmente existem várias ferramentas para o desenvolvimento de portais que se integram com suas implementações de *grids*. Dentre elas pode-se citar: *Grid Portal Development Kit* (GPDK), *Legion Grid Portal*, *Grid Portal Toolkit* (GridPort), *Sun Technical Computing Portal* e *GridSphere*.

No presente trabalho, é realizada a implementação de um portal que possibilita aos usuários, em última análise, a submissão de experimentos relativos à mineração de textos. Contudo, para o seu desenvolvimento, é necessário o domínio de certos conceitos e tecnologias que são brevemente descritos a seguir:

(i) *Servlets*: são classes implementadas em Java que executam uma função específica e geram uma saída em formato HTML. Só podem ser executadas por *web containers*;

(ii) *Portlets*: são componentes web – como *servlets* – especificamente projetados para serem agregados em um contexto de uma página composta. Usualmente, muitos *portlets* são invocados em uma simples chamada a uma página de um portal. Cada *portlet* produz um fragmento de página que é combinado com outros fragmentos de outros *portlets*, onde todos juntos formam uma página do portal;

(iii) *Web Container*: é um servidor que executa *servlets* e permite o acesso destes *servlets* via HTTP. O *web container* mais usado atualmente é o *Apache Tomcat*, que é utilizado

no sistema desenvolvido nesta dissertação;

(iv) *Portlet Container*: é um *software* rodando como uma aplicação Java, dentro de um *web container*, que permite a um *portlet* ser executado;

(v) *Simple Object Access Protocol* (SOAP): é um protocolo leve para troca de informações. Parte da sua especificação é composta por um conjunto de regras que descrevem como utilizar o XML para representar os dados. Outra parte define o formato de mensagens, convenções para representar as chamadas de procedimento remoto (RPC) utilizando o SOAP, e associações ao protocolo HTTP;

(vi) *Security Assertion Markup Language* (SAML): é um *framework* para troca de informações seguras relativas a autenticação e autorização entre duas entidades na rede usando SOAP;

(vii) *Web Services*: um *web service* é um componente, ou unidade lógica de aplicação, acessível através de protocolos padrão de Internet. Como componentes, esses serviços possuem uma funcionalidade que pode ser reutilizada sem a preocupação de como é implementada. O modo de acesso é diferente de alguns modelos anteriores, onde os componentes eram acessados através de protocolos específicos, como o DCOM, RMI ou IIOP. *Web Services* combinam os melhores aspectos do desenvolvimento baseado em componentes e a *Web*;

(viii) *Web Services Definition Language* (WSDL): é uma linguagem baseada em XML, com a finalidade de documentar as mensagens que o *Web service* aceita e gera. Esse mecanismo padrão facilita a interpretação dos contratos pelos desenvolvedores e ferramentas de desenvolvimento;

(ix) *Universal Description, Discovery, and Integration* (UDDI): também é necessária uma forma de localização dos *Web services*. O protocolo UDDI define um formato para o documento e um protocolo para devolver esse documento, possibilitando a localização dos serviços em um *web site* conhecido. No entanto, é comum que não se saiba as URLs onde os serviços podem ser encontrados. O UDDI é um mecanismo para os fornecedores anunciarem a existência de seus serviços e para os consumidores localizarem os serviços de seu interesse;

(x) *Web Services for Remote Portlets* (WSRP): define um padrão para *web services* interativos e orientados à apresentação. Habilita o *web server* a retornar trechos de código HTML. A criação do portal torna-se uma simples agregação destes trechos de HTML, requerendo, desta maneira, teoricamente, pouco esforço de programação;

(xi) *Service Oriented Architecture* (SOA): é uma arquitetura em que um serviço é definido como uma função de negócio sem estado e auto-contida, que aceita requisições através de interfaces padronizadas e interoperáveis, executa uma unidade de trabalho do processo de negócio e retorna resultados. Tudo isto feito com baixo acoplamento e altíssima interoperabilidade;

(xii) *Java Commodity Grid* (COG) *Toolkit*: é um conjunto de classes Java que definem uma camada de abstração entre uma aplicação Java e o *middleware* Globus. No presente trabalho todos os acessos ao *grid* NACAD são realizados utilizando esta camada de abstração. Na figura 3.9 são ilustradas as camadas de acesso ao grid deste trabalho.

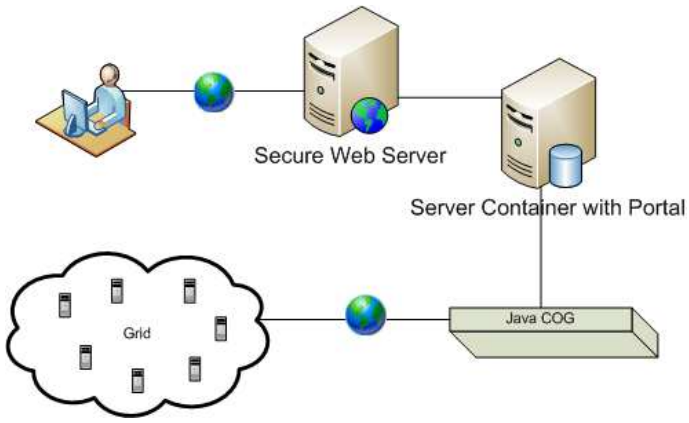


Figura 3.9: Camadas de Acesso ao *Grid* NACAD

Na presente dissertação não foi utilizada nenhuma ferramenta específica para a construção do portal, optando-se pela implementação com ferramentas que são descritas no próximo capítulo, porém mantendo as funcionalidades básicas pertinentes a um portal, com base nos conceitos descritos nos itens anteriores.

Capítulo 4

O ambiente Computacional

O ambiente computacional configurado para a utilização do *software* produzido para essa dissertação, envolve ferramentas de código aberto, de maneira a permitir futuras modificações e evoluções que se fizerem necessárias. A seguir são apresentadas as ferramentas e uma breve explanação sobre seus conceitos principais.

4.1 Eclipse

O Eclipse é uma das ferramentas de desenvolvimento de *software* mais populares atualmente para a construção de sistemas em plataforma Java, sendo considerada uma das ferramentas mais importantes em se tratando de iniciativas *open-source* (código aberto). Como IDE (*Integrated Development Environment* - Plataforma de Desenvolvimento Integrada), possui facilidades como visualização de todos os arquivos contidos no projeto de forma clara, ferramentas de gerenciamento de trabalho coletivo, compilação em tempo real, geração automática de código, dentre outras.

Em virtude do uso da tecnologia de *plug-ins*, o Eclipse permite personalizar o ambiente de trabalho do desenvolvedor de acordo com o projeto que está sendo desenvolvido, seja ele um simples projeto com páginas HTML estáticas, até aplicações com uso de EJBs (*Enterprise Java Beans*), *frameworks* diversos ou J2ME (*Java to MicroEdition*) [97]. Além disso, a tecnologia de *plug-in* possibilita ao desenvolvedor a criação de seus próprios *plug-ins*.

A idéia de *plug-in* está associada a uma extensão das funcionalidades da ferramenta, ou seja, "plugar" um fragmento de *software* ao que já existe, que possibilitará que o desenvolvedor realize atividades além das que já eram possíveis. Com isso, pode-se adicionar recursos

a um ambiente, criar novos ambientes de programação para outras linguagens ou para um propósito específico [98].

Como o termo *open-source* há anos deixou de ser sinônimo de ferramentas sem recursos, com *bugs* e sem suporte algum, o Eclipse permite que se faça em seu ambiente o mesmo que poderia ser feito em ferramentas comerciais, como o JBuilder [99].

Atualmente, o responsável pela continuidade do desenvolvimento do Eclipse é o Consórcio Eclipse.org [100], criado pela IBM, empresa responsável pelo desenvolvimento desta ferramenta em sua fase inicial e depois disponibilizada como projeto *open-source*. Hoje o consórcio é formado por grandes empresas de tecnologia de *software* e desde o ano de 2004, tornou-se independente, sem a influência direta da IBM [100].

O diferencial do Eclipse é a flexibilidade proporcionada ao desenvolvedor. Ele sempre trabalha em um *workbench*, isto é, um ambiente que pode ser configurado conforme suas necessidades.

4.2 Java

Java é uma linguagem computacional completa, adequada ao desenvolvimento de aplicações baseadas na Internet, Intranet ou ainda programas *stand-alone* [101].

Foi desenvolvida na 1ª metade da década de 1990 nos laboratórios da *Sun Microsystems* com o objetivo de ser mais simples e eficiente do que suas predecessoras. O alvo inicial era a produção de *software* para produtos eletrônicos de consumo (fornos de microondas, agendas eletrônicas etc.). Um dos requisitos para esse tipo de *software* é ter código compacto e de arquitetura neutra.

A linguagem obteve sucesso em cumprir os requisitos de sua especificação, mas apesar de sua eficiência, não conseguiu sucesso comercial. Com a popularização da Internet, os pesquisadores da *Sun Microsystems* perceberam que aquele seria um nicho ideal para aplicar a recém criada linguagem de programação. Assim, adaptaram o código Java para que pudesse ser utilizado em microcomputadores conectados à Internet, mais especificamente no ambiente da *World Wide Web*. O Java permitiu a criação de programas batizados de *applets*, que trafegam e trocam dados através da Internet e utilizam a interface gráfica de um *web browser*. Os pesquisadores da *Sun Microsystems* também implementaram o primeiro *browser* compatível com o Java, o *HotJava*, que fazia a interface entre as aplicações Java e o

sistema operacional dos computadores.

Com isso, a linguagem atingiu uma elevada popularidade, passando a ser usada amplamente na construção de documentos *web* que permitem maior interatividade. Os principais *web browsers* disponíveis comercialmente passaram a suportar programas em Java. Outras tecnologias em áreas como computação gráfica e banco de dados, também foram integradas com o novo paradigma proposto pela linguagem Java: aplicações voltadas para o uso de redes de computadores.

Gosling e McGilton [102] destacaram, pela primeira vez, as características gerais de Java:

(i) Simplicidade e eficiência de código orientado a objetos;

(ii) Código interpretado e portável;

(iii) Segurança;

(iv) Aplicações distribuídas e processamento paralelo.

Sem dúvida, atualmente, o Java é a linguagem padrão para o desenvolvimento de sistemas para a Internet.

4.3 Java Server Pages (JSP)

O JSP é uma tecnologia para desenvolvimento de aplicações *web* que tem a vantagem de ser multi-plataforma, como também é de fácil codificação, facilitando assim a construção e manutenção de uma aplicação. Esta tecnologia permite separar a programação lógica (parte dinâmica) da programação visual (parte estática), possibilitando o desenvolvimento de aplicações mais robustas, em que o programador e o *designer* podem trabalhar no mesmo projeto, porém, de maneira independente.

JSP utiliza *tags* (que são comandos que são executados e que são identificados por algum separador) semelhantes ao XML, que encapsulam a lógica de programação que gera o conteúdo das páginas. Além disso, a lógica de aplicação pode residir em recursos baseados no servidor (por exemplo, arquitetura de componentes Java *Beans*), que a página acessa através das *tags*. Toda a formatação (HTML ou XML) é passada diretamente para a página de resposta.

É necessário ter um servidor de aplicação, para se executar uma página JSP, que receba a solicitação do usuário, efetue o processamento e envie a resposta ao mesmo. Existem vários servidores de aplicação disponíveis na *web* (*software* livre), dentre eles o *Jakarta Tomcat*, JBoss etc. Nesta dissertação o *Tomcat* é usado como servidor de aplicações.

Na figura 4.1 é ilustrado o funcionamento da requisição de uma página JSP.

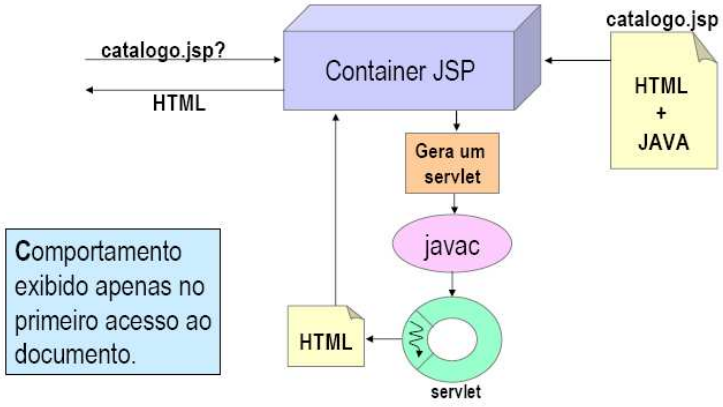


Figura 4.1: Funcionamento de uma página JSP

4.4 XML

XML ou (*eXtensible Markup Language*, ou Linguagem de Marcação Estendida) pode ser definida como uma linguagem de marcação de dados extensível – ao contrário do HTML – que foi projetada para permitir o uso do SGML (*Standard Generalized Markup Language*) na *World Wide Web*. Ela provê um formato para descrever dados estruturados que facilita declarações mais precisas do conteúdo.

O XML não é uma simples linguagem de marcação pré-definida: ela é uma metalinguagem – uma linguagem usada para descrever outras linguagens – que permite que o seu usuário defina a sua própria marcação. Uma linguagem de marcação pré-definida como o HTML especifica um modo de descrever informação em apenas uma classe específica de documento. O XML permite que o usuário defina as suas próprias linguagens de marcação para atender à inúmeras classes de documentos diferentes. Isto é possível porque o XML é escrito em SGML, a metalinguagem padronizada internacionalmente para sistemas de marcação de texto.

4.5 Padrões de Projeto

Segundo Gamma et al. [103], "um padrão de projeto (*design pattern*) é uma descrição de comunicação de objetos que são personalizados para resolver um problema genérico em um contexto particular."

Pode-se entender que um *pattern* é uma solução para problemas de modelagem que ocorrem com freqüência em situações específicas. Mais especificamente, um *pattern* é um par nomeado problema/solução que pode ser aplicado em novos contextos, com conselhos sobre sua aplicação em novas situações e uma discussão sobre as conseqüências de seu uso [18].

Documenta-se um *Design Pattern* através de algumas características essenciais, que tipicamente são descritas por:

(i) Nome do *Pattern*;

(ii) Problema: descreve a situação enfrentada pelos desenvolvedores;

(iii) Solução: forma de modelagem independente de cenário específico;

(iv) Conseqüências: ganhos na implementação e resultados esperados.

Os *Design Patterns* são classificados em três categorias:

(i) Comportamental: voltados para a modelagem da comunicação entre os objetos e também o fluxo em cenários complexos;

(ii) Criação: *patterns* que criam objetos sem que se tenha que instanciar a classe diretamente;

(iii) Estrutura: ajudam a compor um grupo de objetos em largas estruturas.

Design Patterns oferecem uma metodologia simples para a implementação de soluções, garantindo a qualidade do produto desenvolvido, pois comprovadamente, expressam, de fato, a melhor prática para a resolução de um determinado problema. Algumas das vantagens citadas por Gamma et al. [103] são:

(i) Facilitam a reutilização de um projeto;

(ii) Ajudam a escolher alternativas de projetos que tornam um sistema reutilizável, evitando alternativas que comprometam a reutilização;

(iii) Melhoram a documentação e a manutenção de sistemas ao fornecer uma especificação explícita de interações de classes e objetos subjacentes;

(iv) Diminuem o tempo de desenvolvimento do sistema.

Os padrões utilizados na modelagem do sistema desta dissertação foram desenvolvidos principalmente visando o fator manutenibilidade, uma vez que esta aplicação vai sofrer alterações ao longo do tempo, com a inclusão de novos algoritmos.

Os padrões utilizados seguem abaixo, e são detalhados por Gamma et al. [103]:

(i) *Singleton*: garante que uma classe tenha somente uma instância e fornece um ponto global de acesso a ela.

(ii) *Command*: encapsula uma solicitação como um objeto, permitindo que clientes sejam parametrizados com diferentes solicitações, que solicitações sejam enfileiradas ou registradas (*log*) e que suporte operações que possam ser desfeitas.

(iii) *Adapter*: converte a interface de uma classe em outra interface esperada pelos clientes. O *Adapter* permite que certas classes trabalhem em conjunto, pois de outra forma seria impossível por causa de suas interfaces incompatíveis.

(iv) *Factory Method*: define uma interface para criar um objeto, mas deixa as subclasses decidirem qual classe será instanciada. O *factory method* permite a uma classe postergar a instanciação às subclasses.

4.6 Arquitetura MVC

A arquitetura MVC (*Model, View, Controller*) tem como estratégia de desenvolvimento a divisão de uma aplicação em três camadas: *Model* (Modelo), *View* (Visão) e *Controller* (Controlador ou Controle). Segundo Gamma et al. [103], antes do padrão MVC os projetos agrupavam essas camadas. No entanto, com o intuito de aumentar a flexibilidade e a reutilização dos sistemas, com o advento do MVC essas camadas foram separadas, isolando a camada de visão da camada de modelo, através de um protocolo para inserção/notificação

(*subscribe/notify*) entre estas camadas. Esse protocolo garante que uma visão reflita o estado atual do modelo, ou seja, quando o modelo sofrer uma modificação, esse protocolo deve comunicar as alterações feitas para as visões que estão fazendo acesso a ele. Ao utilizar essa arquitetura, os projetos proporcionam uma maior confiabilidade permitindo o desenvolvimento de produtos reutilizáveis e de fácil manutenção. Nas Figuras 4.2 e 4.3 são ilustradas duas visões para o funcionamento do MVC.

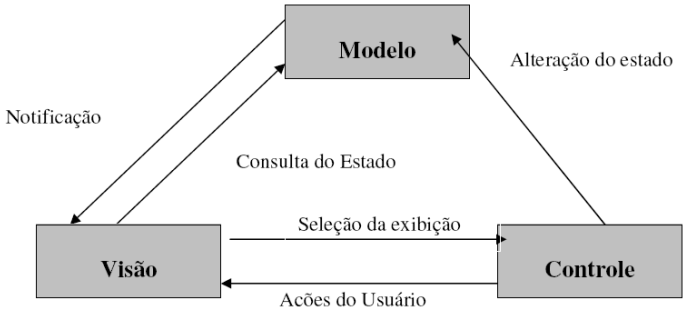


Figura 4.2: Funcionamento do MVC

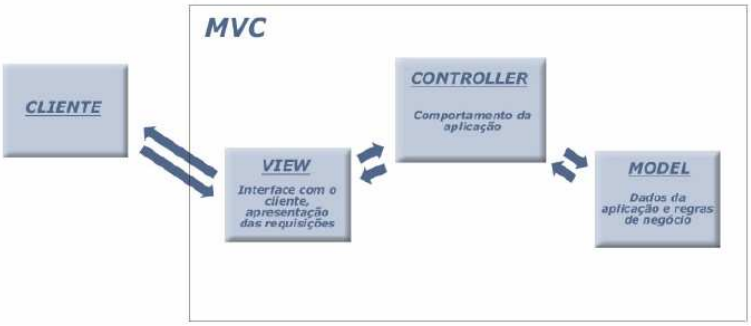


Figura 4.3: Funcionamento do MVC: outra visão

Analisando a figura 4.2, observa-se que todas as ações executadas pelo usuário são gerenciadas pela camada controle. Quando há uma alteração no modelo, o mesmo deve notificar as visões que estão conectadas a ela. Após a notificação, a camada visão deve consultar o modelo a fim de recuperar o estado atual do mesmo. A seguir são apresentados alguns detalhes de cada uma destas camadas:

(i) Visão: camada responsável por abrigar a interface gráfica da aplicação. Promove o acesso e modificação dos dados contidos na camada modelo. É a camada de interface do usuário, que recebe as entradas dos dados e exibe os resultados;

(ii) Modelo: camada responsável por abrigar a lógica do negócio, os dados fundamentais da aplicação e as regras do negócio que controlam o acesso e modificações feitas nesses

dados. Esta camada modela os dados da aplicação e o comportamento por trás do processo de negócio, além de controlar o armazenamento dos dados e gerenciar a criação de novos dados;

(iii) Controle: camada responsável por atender às solicitações realizadas pelo usuário através da camada visão. É esta camada que recebe as requisições feitas pelo usuário, faz a interpretação dessa requisição para acessar a camada modelo e realiza as alterações.

4.7 Arquitetura Modelo 2

Basicamente, duas tecnologias estão envolvidas para o desenvolvimento de aplicações *web*, tendo como base a linguagem Java. A primeira delas são os *servlets* e a segunda é a linguagem JSP, conforme definido na seção 4.3. A linguagem JSP está diretamente ligada ao modelo MVC, também chamado de Modelo 1. Já o modelo 2 está ligado à tecnologia de *servlets* e à linguagem JSP.

Basicamente, no Modelo 1 é definida a utilização de páginas JSP. Os programas produzidos baseados neste modelo são compostos por entidades de negócio e de interface que se entrelaçam. As aplicações desenvolvidas conforme esse modelo são de difícil manutenção, reutilização e o trabalho em equipe fica comprometido, pois não há uma separação do código entre as camadas.

Na arquitetura Modelo 2 as tecnologias *servlet* e JSP são utilizadas no desenvolvimento de aplicações *web*. Nesta arquitetura as páginas JSP são responsáveis por apresentar os dados aos usuários, e os *servlets* ficam encarregados de acessar os dados e controlar o fluxo navegacional.

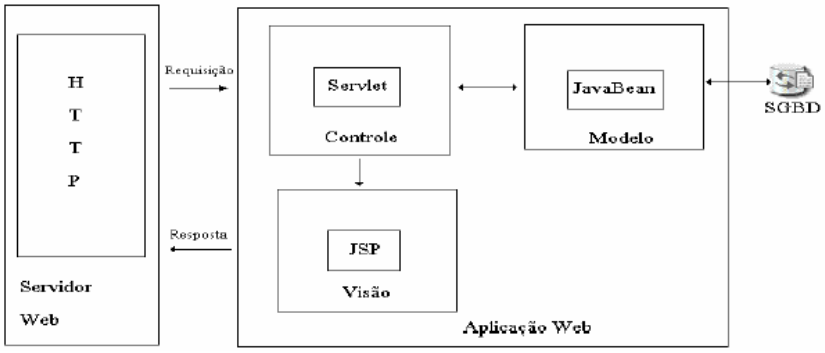


Figura 4.4: Funcionamento da arquitetura Modelo 2

Como pode ser observado na figura 4.4, onde é ilustrado o funcionamento do Modelo 2, o servidor *web* direciona todas as requisições do cliente para um *servlet* existente na camada de controle, que é a responsável por acessar as classes da camada modelo e apresentar a resposta a essa requisição por meio de uma página JSP da camada de visão. Desta maneira, futuras manutenções no *software* podem ser realizadas com menos esforço, assim como os componentes de *software* produzidos podem ser reutilizados, devido à evidente independência entre as camadas.

4.8 *Frameworks*

Um *framework* é um conjunto de componentes (classes e interfaces) que funcionam juntos para solucionar um determinado problema de *software*. Segundo Johnson e Foote [104], um *framework* é uma aplicação reutilizável e semi-completa que pode ser especializada para produzir aplicações personalizadas. A principal finalidade de um *framework* é proporcionar a sua reutilização em outros projetos, diminuindo a complexidade na implementação de *softwares*. Segundo Gamma et al. [103], um *framework* determina a arquitetura de sua aplicação, ou seja, define a estrutura geral, sua divisão em classes e objetos e, conseqüentemente, as responsabilidades entre si, assim como a forma de colaborarem e o fluxo de controle. Um *framework* possui as seguintes características:

(i) É composto por múltiplas classes ou componentes, cada um devendo prover uma abstração de um conceito particular;

(ii) Define como as abstrações trabalham juntas para resolver um problema;

(iii) Seus componentes são reutilizáveis;

(iv) Organiza padrões em alto nível.

Existem vários *frameworks* disponíveis, como, por exemplo: Struts, WebWork, Spring, entre outros. Todos estes são baseados na arquitetura MVC. Dentre os *frameworks* citados, destaca-se o Struts, pois é o mais utilizado pelos projetistas de aplicações *web* Java devido à sua simplicidade, recursos oferecidos e à grande quantidade de material disponível para pesquisa.

4.9 *Struts Framework*

O *Struts* é um *framework* que foi idealizado por Craig McClanahan em maio de 2000. A organização responsável por sua atualização é a *Apache Software Foundation*. Este *framework* é *open-source* e de domínio público e, tem como finalidade prover uma estrutura para o desenvolvimento de aplicações *web*, usando *Servlets* e JSP.

No *Struts* é fornecido um componente controlador chamado *ActionServlet* para controlar o fluxo navegacional da aplicação. O controlador é auxiliado por outros componentes, tais como *ActionMappings*, *Action*, *ActionForm*, *ActionForward*, entre outros, para acessar os dados, sejam eles provenientes de uma requisição HTTP ou de um Sistema de Gerenciamento de Banco de Dados (SGBD) na camada modelo. Outra característica é que bibliotecas de *tags* são disponibilizadas para apresentação, captação e manipulação de dados nas páginas JSP, como também *frameworks* que facilitam algumas tarefas de implementação, como validação de dados e criação do *layout* das páginas.

Utilizando o *Struts*, é possível desenvolver páginas internacionalizadas, ou seja, em vários idiomas, apenas criando e mantendo arquivos com extensão ".properties", que contém todos os textos, *labels* e mensagens apresentadas em uma página.

Com o *Struts* são disponibilizados os componentes necessários para implementação da camada controle, sendo integrado com outras tecnologias, tais como JSP, bibliotecas de *tags* (*Tag Libraries*), *JavaBeans*, JDBC, entre outras. Estas tecnologias são usadas nas camadas dando suporte ao *Struts*, onde o JSP e as bibliotecas de *tags* são usados na camada visão para inserção e apresentação dos dados. Já o *JavaBeans* é utilizado na camada modelo para gerenciar os dados da aplicação. Portanto, para aplicar o *Struts* no desenvolvimento de uma aplicação *web* é necessário ter um mínimo de conhecimento dessas tecnologias.

O *Struts* é uma implementação do *design pattern* MVC (*Model-View-Controller*) para aplicações Java com Internet. O objetivo do *pattern* MVC, como já foi dito anteriormente, é separar de maneira clara a camada de visão (*View*) da camada de Negócio (*Model*), como ilustrado na figura 4.5.

As etapas do fluxo de navegação do *Struts*, ilustrado na figura 4.6 é descrito como segue:

(1) O usuário faz uma solicitação através de uma *URL* no *browser*, por exemplo

http://localhost:8080/cadastro/listUsers.do

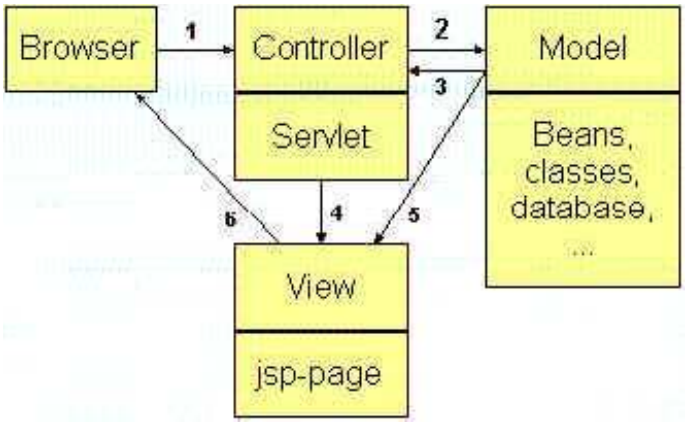


Figura 4.5: O Padrão MVC

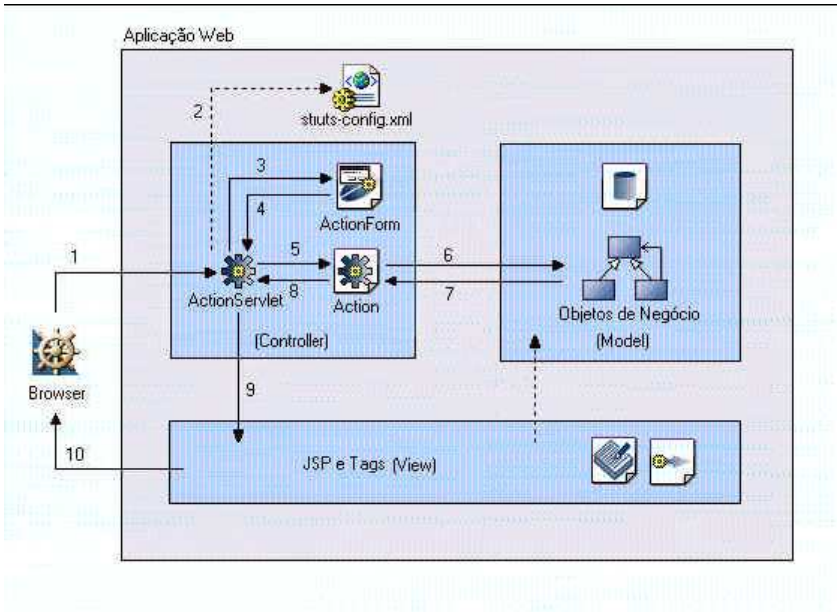


Figura 4.6: Fluxo de navegação nos componentes do Struts

Note que a terminação da *URL* é um ".do", que será usado para invocar (na verdade mapear) o *servlet controller* do *Struts*.

(2) Se for a primeira solicitação que o *container* recebeu para esta aplicação, ele invocará o método *init()* da *ActionServlet* (*controller do Struts*) e carregará as configurações do arquivo *struts-config.xml* em estruturas de dados na memória. Vale lembrar que esta passagem só será executada uma única vez, pois nas solicitações subsequentes, o *servlet* consulta estas estruturas na memória para decidir o fluxo a ser seguido.

(3) Baseado no fluxo definido no arquivo *struts-config.xml* e que neste momento já se encontra carregado em estruturas na memória, o *ActionSerlet* identificará qual o *ActionForm* (classe para a validação dos dados) invocará. A classe *ActionForm*, através do

método *validate*, verificará a integridade dos dados que foram recebidos na solicitação proveniente do *browser*.

(4) O controle da aplicação é retomado pelo *ActionServlet*, que confere o resultado da verificação do *ActionForm*. Se faltou alguma informação (campo não preenchido, valor inválido etc), o usuário recebe um formulário HTML (geralmente o mesmo que fez a solicitação), informando o motivo do não atendimento da solicitação, para que o usuário possa preencher corretamente os dados e fazer uma nova solicitação. Se não faltou nenhuma informação, ou seja, todos os dados enviados corretamente, o *controller* passa para o próximo passo (*Action*).

(5) O *ActionServlet*, baseado no fluxo da aplicação (estruturas já carregadas em memória) invoca uma classe *Action*. A classe *Action* passará pelo método *execute* que delegará a requisição para a camada de negócio.

(6) A camada de negócio executará algum processo (geralmente popular um *bean*, ou uma coleção). O resultado da execução deste processo (objetos já populados) será usado na camada de apresentação para exibir os dados.

(7) Quando o controle do fluxo da aplicação voltar ao *Action* que invocou o processo da camada modelo (*model*), será analisado o resultado e definido qual o mapa adotado para o fluxo da aplicação. Neste ponto, os objetos que foram populados na camada de negócio serão "anexados" como atributos na seção do usuário.

(8) Baseado no mapeamento feito pelo *Action*, o *Controller* faz um direcionamento para o JSP para apresentar os dados.

(9) Na camada visão (*View*), os objetos que foram definidos como atributos da sessão do usuário serão consultados para montar o arquivo HTML para o *browser*.

(10) O arquivo HTML da resposta requisitada pelo usuário é recebido.

4.10 Linguagem de Modelagem Unificada

A Linguagem de Modelagem Unificada ou UML é uma linguagem-padrão para a estruturação de projetos de *software*. Sua abrangência vai desde a modelagem de sistemas de

informação corporativos a serem distribuídos em aplicações voltadas para a *web*, até sistemas complexos embutidos de tempo real.

Para cumprir seu objetivo, a UML permite que seus usuários modelem um sistema sob diferentes perspectivas. Cada uma destas perspectivas é uma abstração apresentada por diagramas criados a partir dos recursos oferecidos pela linguagem de modelagem.

Em UML, a criação destes diagramas envolve a identificação de itens que formam o vocabulário do sistema e a especificação de como estes itens relacionam-se entre si.

No presente trabalho são usados os diagramas de Casos de Uso, o diagrama de Classes e o diagrama de Pacotes, que são detalhados nas próximas seções.

4.11 O Processo Unificado de Desenvolvimento de Software

Segundo Jacobson et al. [105], um processo é um conjunto de passos que define quem está fazendo o que, quando e como, para alcançar determinado objetivo. Na engenharia de *software* este objetivo é entregar de maneira eficiente e previsível, um produto capaz de atender às necessidades de seu negócio.

A UML por ser apenas uma linguagem para modelagem orientada a objetos e amplamente independente de processo, indica somente como criar e ler modelos bem-formados, mas não aponta quais modelos serão criados nem quando deverão ser criados. Essa tarefa cabe ao processo de desenvolvimento de sistemas. A UML é, portanto, apenas parte de um método para o desenvolvimento de sistemas e, o fato de ser independente de processo, permite que possa ser utilizada com vários processos de engenharia de *software*.

Porém, a adoção do Processo Unificado (PU), como o processo responsável pelo desenvolvimento de sistemas modelados em UML, é mais adequada, tendo em vista que este está diretamente ligado à UML, utilizando-a como notação para uma série de modelos.

O PU captura algumas das melhores práticas atuais de desenvolvimento de *software*, de maneira que pode ser adaptado a uma ampla variedade de projetos. Porém, o PU não pode ser considerado como apenas uma união das melhores e principais características das metodologias mais populares, ele também possui características específicas, como ser orientado por casos de uso, ser centrado na arquitetura, ser iterativo e incremental. Na figura 4.7 é ilustrado o Processo Unificado.

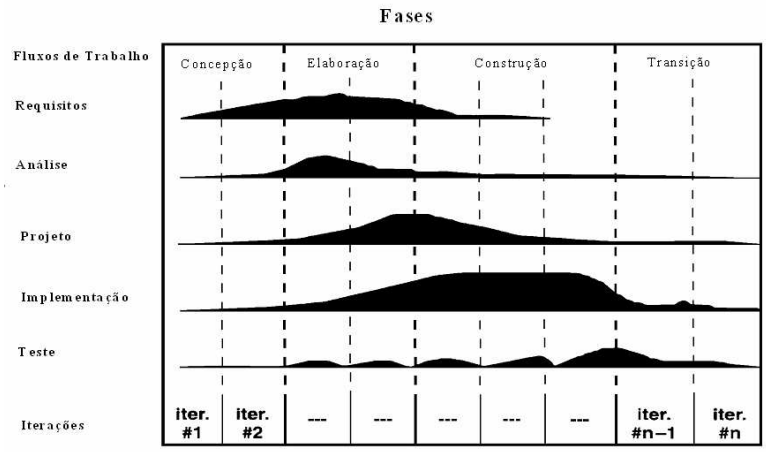


Figura 4.7: O Processo Unificado

4.11.1 Características

São três as principais características do PU: ser orientado por casos de uso, ser centrado na arquitetura e ser iterativo e incremental. Estas três características se relacionam entre si e são igualmente importantes. A arquitetura fornece a estrutura que guia o trabalho de cada iteração; os casos de uso definem as metas e dirigem o trabalho de cada iteração. Remover uma das três características reduziria drasticamente o valor do Processo Unificado. Estas três características funcionam como as bases do PU. Se uma dessas bases for retirada, o processo é prejudicado.

4.11.1.1 Processo Orientado por Casos de Uso

Um caso de uso é uma seqüência de ações de um sistema que retorna ao usuário um resultado de valor. Um conjunto de casos de uso, definido sob determinado contexto, forma o diagrama de casos de uso que descreve a funcionalidade do sistema sob este contexto. Em outras palavras, um diagrama de casos de uso define a funcionalidade de um sistema para cada usuário.

Um processo orientado por casos de uso faz com que um sistema seja desenvolvido sob a perspectiva de atender, especificamente, às necessidades de cada usuário que interage com o mesmo, evitando, assim, que o sistema possa ser desenvolvido com funcionalidades desnecessárias. Entretanto, casos de uso não são ferramentas ligadas apenas à especificação de requisitos do sistema, mas também ao seu projeto, implementação e testes, ou seja, o processo de desenvolvimento do sistema é orientado por casos de uso.

Ser orientado por casos de uso significa que o processo de desenvolvimento segue um fluxo, ou seja, o processo passa por uma série de fluxos de trabalho que derivam dos casos de uso.

Desta maneira, os modelos de análise, projeto e implementação são criados pelos desenvolvedores com base no modelo de casos de uso. Este processo é conhecido como realização de casos de uso e é evidenciado durante todo o ciclo de vida do PU.

Casos de uso são desenvolvidos de acordo com a arquitetura do sistema. Assim, através deles é definida a arquitetura do sistema , e esta, por sua vez, influencia a seleção dos casos de uso. Conseqüentemente, tanto a arquitetura do sistema quanto os casos de uso, evoluem durante o ciclo de vida do sistema.

4.11.1.2 Processo Centrado na Arquitetura

Segundo Jacobson et al. [105], a arquitetura é a visão de todos os modelos, que juntos representam o sistema com um todo. O conceito de arquitetura de *software* engloba os aspectos estáticos e dinâmicos mais significativos do sistema.

De qualquer modo, a arquitetura também é influenciada por muitos outros fatores, como a plataforma na qual o *software* será implantado, os blocos de construção reutilizáveis, requisitos funcionais e não-funcionais.

A arquitetura é uma visão do projeto do sistema como um todo, destacando suas características mais importantes, mas sem entrar em detalhes. Segundo Jacobson et al. [105], o que é significante em um sistema depende do ponto de vista de cada desenvolvedor e que muitas vezes uma opinião sensata está ligada à experiência. Portanto, o valor da arquitetura depende das pessoas que estão ligadas ao desenvolvimento do sistema. De qualquer modo, o processo ajuda o analista a concentrar-se nas metas corretas, como intelegibilidade, poder de recuperação para mudanças futuras e reutilização.

A relação existente entre casos de uso e a arquitetura é que os casos de uso estão ligados à funcionalidade de um sistema e a arquitetura, por sua vez, está ligada à forma deste. Além disso, funcionalidade e forma devem estar balanceadas para se alcançar um produto final de qualidade, ou seja, casos de uso e arquitetura devem estar ligados a tal ponto que o primeiro seja desenvolvido de acordo com a arquitetura e, esta por sua vez, forneça um ambiente para a realização de todos os requisitos dos casos de uso.

Assim, a arquitetura de um sistema deve ser projetada de maneira a permitir que o sistema evolua, não apenas durante o início de seu desenvolvimento, como também em suas versões futuras. Para isso, os analistas devem dar atenção especial aos casos de uso de maior complexidade. Estes representam de 5 a 10% do total dos casos de uso, porém são os que representam mais riscos para o sistema e devem ser solucionados o mais rápido possível.

4.11.1.3 Processo Iterativo e Incremental

Durante o desenvolvimento de um sistema, é interessante dividi-lo em mini-projetos. Cada mini-projeto é uma iteração que resulta em um incremento. Iterações referem-se às etapas do fluxo de trabalho, e incrementos ao avanço no desenvolvimento do produto. Para serem mais efetivas, as iterações devem ser controladas, ou seja, devem ser executadas de modo planejado. Por isso são consideradas mini-projetos.

Os desenvolvedores baseiam-se em dois fatores para definir o que será implementado em uma iteração. Primeiro, a iteração lida com um grupo de casos de uso que juntos representam o funcionamento do sistema em desenvolvimento. Segundo, a iteração lida com os riscos mais significativos do empreendimento. Iterações sucessivas constroem um conjunto de artefatos a partir do estado em que estes foram deixados ao término da iteração passada. Assim, partindo dos casos de uso, o mini-projeto prossegue através dos fluxos de trabalho subseqüentes (análise, projeto, implementação e teste), até alcançar a forma de código executável.

Em cada iteração, os desenvolvedores identificam e especificam os casos de uso relevantes, criam o projeto de acordo com a arquitetura escolhida, implementam o projeto em componentes e verificam se os componentes satisfazem os casos de uso. Se uma iteração alcançar seu objetivo, o desenvolvimento prossegue com a próxima iteração. Porém, se uma iteração não alcançar seu objetivo, os desenvolvedores devem revisar as decisões tomadas anteriormente e tentar uma nova abordagem.

Para conseguir maior economia no desenvolvimento, a equipe de projeto tenta selecionar apenas as iterações necessárias para alcançar as metas predefinidas. Um projeto bem sucedido avança com o mínimo de desvios do curso planejado inicialmente pelos desenvolvedores. Por outro lado, o surgimento de problemas imprevistos acarreta o aumento do número de iterações ou alteração na seqüência das mesmas, fazendo com que o processo de de-

senvolvimento exija mais tempo. Minimizar a possibilidade de surgimento de problemas imprevistos é uma das metas da redução de riscos.

Um processo iterativo controlado tem várias vantagens como:

(i) Iterações controladas reduzem o risco de custo para as despesas em um único incremento. Se os desenvolvedores precisarem repetir a iteração, será perdido apenas o esforço dedicado a uma iteração, não ao produto como um todo;

(ii) Iterações controladas reduzem o risco de violação de prazos. O fato de identificar problemas no início do desenvolvimento, permite que os desenvolvedores aloquem tempo para resolvê-los sem extrapolar prazos;

(iii) Iterações controladas aceleram o tempo de desenvolvimento do projeto como um todo, pelo fato de desenvolvedores trabalharem de maneira mais eficiente baseados em resultados de escopo pequeno e claro. Iterações controladas reconhecem uma realidade muitas vezes ignorada, que é a necessidade dos usuários e requisitos correspondentes não poderem ser completamente definidos no início do desenvolvimento. Eles devem ser refinados em sucessivas iterações. Este modo de operação torna mais fácil a adaptação do sistema a mudanças dos requisitos.

4.12 Modelagem do Sistema

A complexidade dos sistemas atuais é a maior dificuldade encontrada por profissionais encarregados pela manutenção desses sistemas. Em muitos casos, a não obediência de diretrizes provenientes de uma forma de planejamento bem estruturada e que permita a simplicidade na descrição de um sistema, faz com que a utilização e a realização de futuras modificações no mesmo represente um sério obstáculo para a atividade de manutenção. Tais diretrizes devem ser cumpridas através de um modelo conceitual que expresse de maneira clara e precisa o funcionamento do sistema, a fim de evitar o decorrente aumento de sua complexidade.

A implementação da presente dissertação segue os conceitos preconizados pelo Processo Unificado de desenvolvimento de *software*, que determina, na atualidade, as melhores práticas de análise de sistemas.

4.12.1 Diagrama de Casos de Uso

Como foi detalhado na seção 4.11.1.1, um caso de uso é uma seqüência de ações de um sistema que devolve ao usuário um resultado de valor. Através do conjunto dos casos de uso define-se as funcionalidades do sistema. Na figura 4.8 é mostrado o diagrama de casos de uso com as principais funcionalidades do sistema.

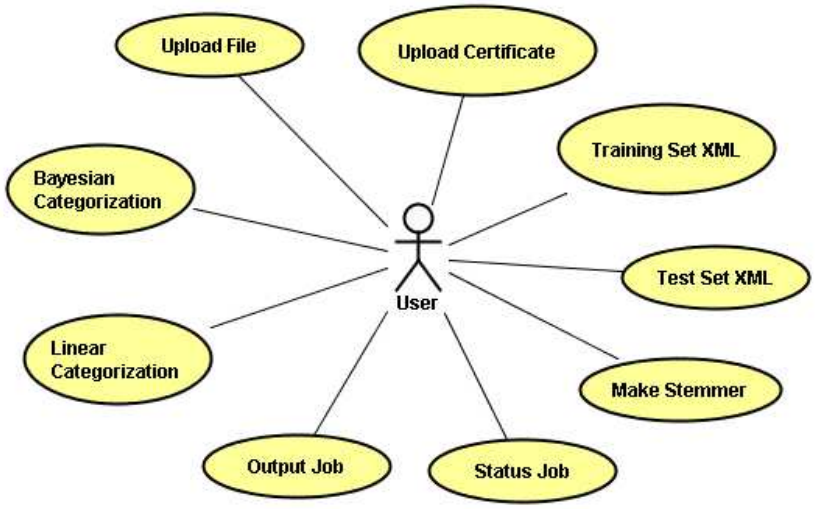


Figura 4.8: Funcionalidades principais do sistema Aîuri

4.12.1.1 Descrição dos Casos de Uso

A seguir são descritos os principais casos de uso.

a) ***Upload certificate***

Ator: Usuário cadastrado

Descrição: Este caso de uso possibilita ao usuário realizar o carregamento de seu certificado para que possa utilizar os recursos do *grid*. O sistema apresenta uma interface para que o usuário selecione o certificado e realize o carregamento.

b) ***Upload file***

Ator: Usuário cadastrado

Descrição: Este caso de uso possibilita ao usuário realizar o carregamento de um arquivo. O sistema apresenta uma interface para que o usuário selecione o arquivo a ser carregado.

c) ***Make Stemmer***

Ator: Usuário cadastrado

Descrição: Este caso de uso gera o arquivo que contem os *stems*. O sistema apresenta uma interface para a seleção dos arquivos cujos termos farão parte do *stemmer*. Após a seleção dos arquivos o usuário deve informar o nome do arquivo de saída que será gerado.

d) ***Training set XML***

Ator: Usuário cadastrado

Descrição: Este caso de uso gera o arquivo contendo os textos do conjunto de treinamento. O sistema apresenta uma interface para a seleção dos arquivos cujos textos farão parte do conjunto de treinamento. Após a seleção dos arquivos o usuário deve informar o nome do arquivo de saída que será gerado, em formato XML, contendo os textos para treinamento.

e) ***Test set XML***

Ator: Usuário cadastrado

Descrição: Este caso de uso gera o arquivo contendo os textos do conjunto de teste. O sistema apresenta uma interface para a seleção dos arquivos cujos textos farão parte do conjunto de teste. Após a seleção dos arquivos o usuário deve informar o nome do arquivo de saída que será gerado, em formato XML, contendo os textos para teste.

f) ***Bayesian categorization***

Ator: Usuário cadastrado

Descrição: Este caso de uso realiza a categorização bayesiana a partir do conjunto de treinamento e devolve o resultado sobre o conjunto de teste. O sistema apresenta uma interface para que o usuário informe os valores das propriedades e selecione o recurso em que o experimento vai ser submetido. Após a submissão o sistema retorna as métricas que foram obtidas.

g) ***Linear categorization***

Ator: Usuário cadastrado

Descrição: Este caso de uso realiza a categorização por ranqueamento linear a partir do conjunto de treinamento e devolve o resultado sobre o conjunto de teste. O sistema apresenta

uma interface para que o usuário informe os valores das propriedades e selecione o recurso em que o experimento vai ser submetido. Após a submissão o sistema retorna as métricas que foram obtidas.

h) ***Status job***

Ator: Usuário cadastrado

Descrição: Este caso de uso realiza a verificação do *status* dos *jobs* submetidos no *grid* EELA. Ao selecionar a opção de verificação do status dos *jobs*, o sistema apresenta uma interface informativa sobre a situação de cada *job*.

h) ***Output job***

Ator: Usuário cadastrado

Descrição: Este caso de uso realiza a busca dos resultados dos *jobs* submetidos no *grid* EELA. Ao selecionar a opção de busca por resultados, o sistema apresenta uma interface com *links* para os arquivos resultantes dos *jobs* submetidos.

4.12.2 Diagrama de Pacotes

Segundo Booch et al. [106], visualizar, especificar, construir e documentar sistemas envolve a manipulação de uma quantidade, por vezes, considerável de classes, interfaces, componentes, nós, diagramas e outros elementos. À medida que os sistemas evoluem, percebe-se a necessidade de organizar esses itens em grupos maiores. Na UML, o pacote é um mecanismo de propósito geral para a organização de elementos da modelagem em grupos.

Os pacotes bem estruturados agrupam elementos que estão próximos semanticamente e que tendem a se modificar em conjunto. Portanto, os pacotes bem estruturados são fracamente acoplados em forte coesão, com acesso altamente controlado ao seu conteúdo [106].

Na figura 4.9 é ilustrado o diagrama de pacotes, que organiza as classes do Projeto Aîuri.

4.12.3 Diagrama de Classes

Segundo Booch et al. [106], as classes são os blocos de construção mais importantes de qualquer sistema orientado a objetos. Uma classe é uma descrição de um conjunto de objetos que compartilham os mesmos atributos, operações, relacionamentos e semântica.

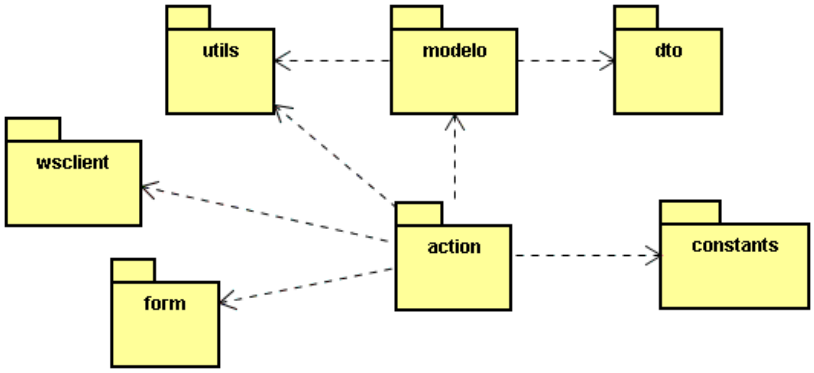


Figura 4.9: Diagrama de pacotes do Projeto Aîuri

Nas figuras 4.10, 4.11, 4.12, 4.13, 4.16 e 4.17 são exibidos os diagramas de classes do Projeto Aîuri.

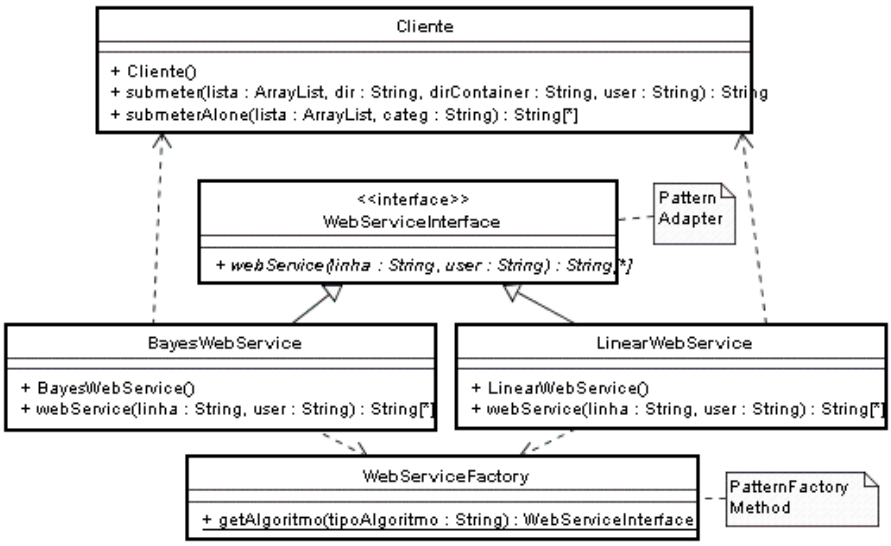


Figura 4.10: Diagrama de Classes do pacote “Modelo” do Projeto Aîuri

No diagrama da figura 4.10 são representadas as classes que implementam os serviços *web*. Estas classes utilizam os padrões de projeto *Adapter* e *Factory Method*. Novos serviços *web* que sejam implementados devem estender a interface *WebServiceInterface*.

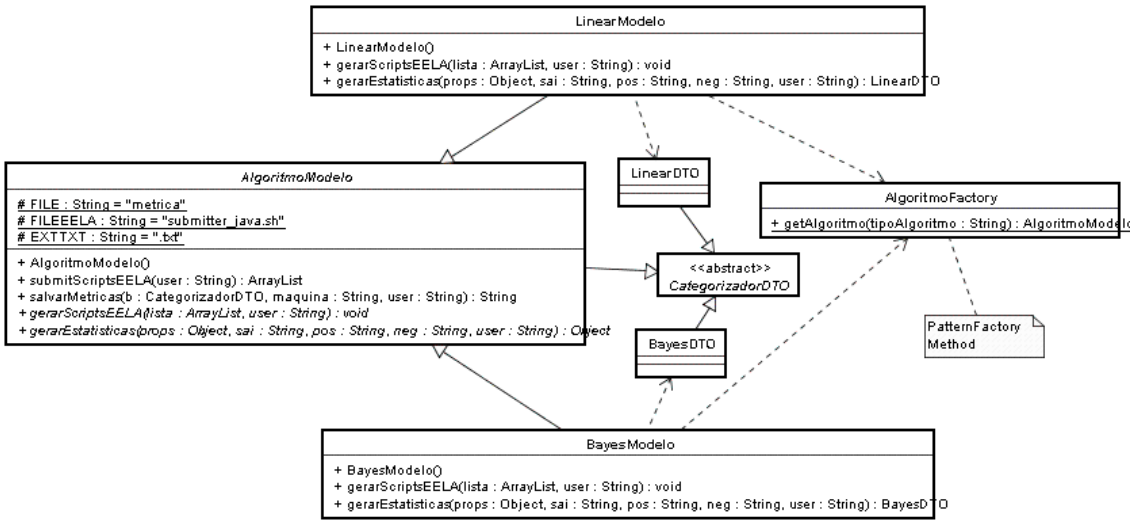


Figura 4.11: Diagrama de Classes do pacote “Modelo” do Projeto Aîuri

No diagrama da figura 4.11 são ilustradas as classes que implementam os algoritmos de categorização. O padrão *Factory Method* também é utilizado. Novos algoritmos que sejam implementados devem estender a classe abstrata *AlgorimoModelo*.

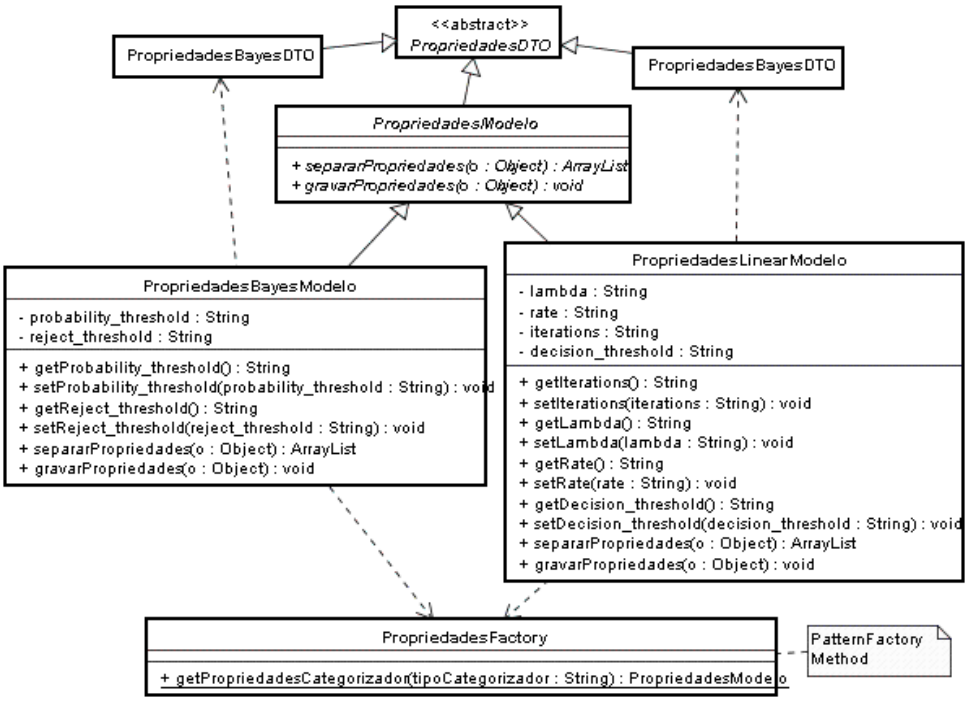


Figura 4.12: Diagrama de Classes do pacote “Modelo” do Projeto Aîuri

O objetivo das classes ilustradas na figura 4.12 é realizar o mapeamento das propriedades referentes aos algoritmos implementados. Os algoritmos de classificação possuem conjuntos de propriedades próprios. Novos algoritmos implementados devem possuir seus respectivos conjuntos de propriedades. Novas propriedades implementadas devem estender a classe abstrata *PropriedadesModelo*.

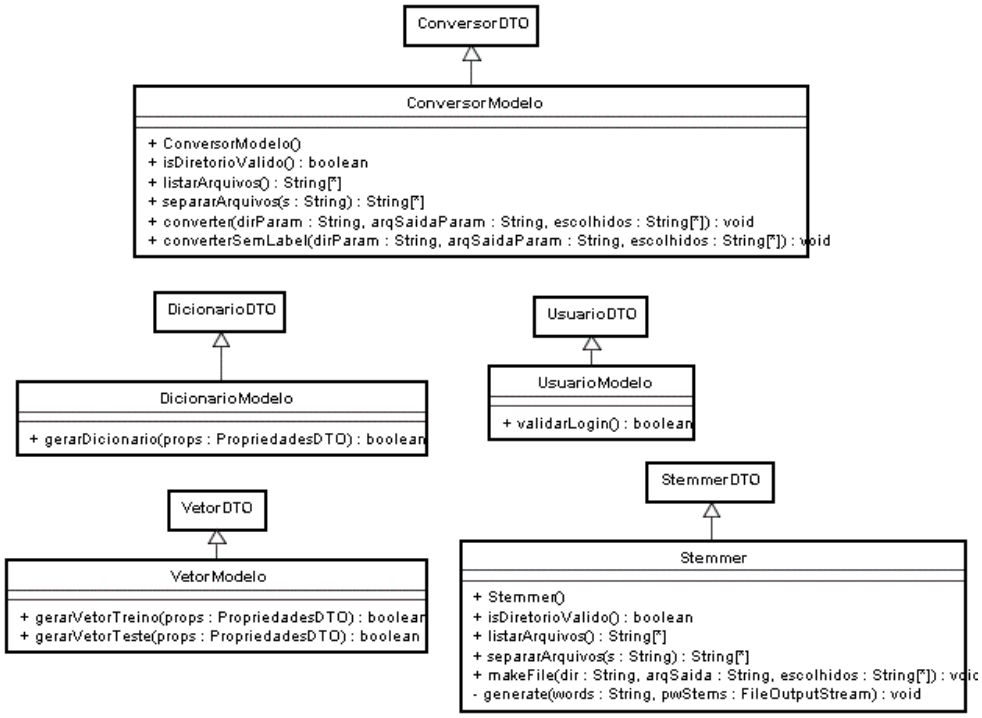


Figura 4.13: Diagrama de Classes do pacote “Modelo” do Projeto Aîuri

Na figura 4.13 são ilustradas as classes necessárias aos algoritmos implementados. A classe *ConversorModelo* possui os métodos que geram os conjuntos de treinamento e teste. A classe *UsuarioModelo* é responsável pela autenticação do usuário no sistema. As classes *DicionarioModelo*, *VetorModelo* e *Stemmer* possuem métodos para a geração do dicionário, dos vetores de treinamento e teste, e do arquivo com os *stems* respectivamente.

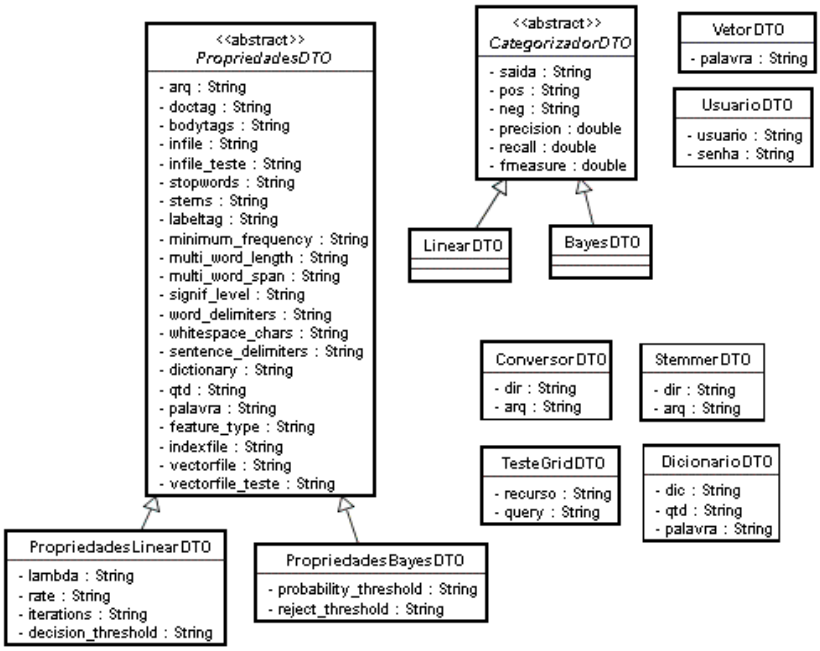


Figura 4.14: Diagrama de classes do pacote “DTO” (Data Transfer Object) do Projeto Aîuri

No diagrama ilustrado na figura 4.16 são definidas as classes do pacote DTO. Estas classes possuem somente atributos, sendo utilizadas para realizar o mapeamento entre a camada de interface e suas respectivas classes da camada modelo, implementadas conforme o padrão MVC.

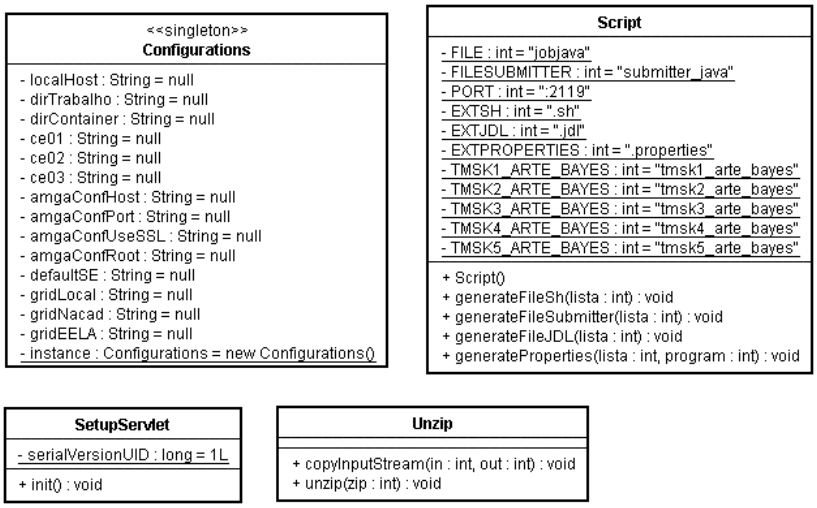


Figura 4.15: Diagrama de classes do pacote “Utils” do Projeto Aîuri

As classes ilustradas no diagrama da figura 4.17 foram implementadas para atender aos requisitos gerais da aplicação. A classe *Configurations* implementa o padrão de projeto *Singleton*. Esta classe contém parâmetros gerais da aplicação. A classe *Script* possui atributos e métodos que são utilizados para a submissão de experimentos no *grid* EELA.

4.13 Funcionamento do Sistema

O portal Aîuri pode ser operado de três formas. A primeira delas, e a mais simples, é o modo “Local”, onde o usuário poderá submeter seus experimentos na própria máquina do portal, sem a necessidade de possuir um certificado para utilização dos serviços de *grid*. A segunda forma, é através da escolha da opção “NACAD”, onde após o *login*, o usuário deverá fazer o carregamento de seu certificado para poder operar em um ambiente de *grid* computacional. Este ambiente opera com os conceitos de serviços *web*, onde o portal, simplesmente, faz chamadas aos serviços *web* que já estão publicados nos nós, passando os parâmetros necessários. Cada novo algoritmo implementado deverá estar publicado nas máquinas do *grid*. O terceiro modo é pela escolha da opção EELA, que é outro ambiente de *grid* computacional. Da mesma maneira que na opção “NACAD”, o usuário deverá fazer o carregamento de seu certificado.

O objetivo do portal Aîuri é possibilitar que os usuários executem seus experimentos em ambientes diversificados.

O sistema deve ser iniciado através do *browser*, onde será apresentada a tela de *login*

conforme mostrado na figura 4.16.

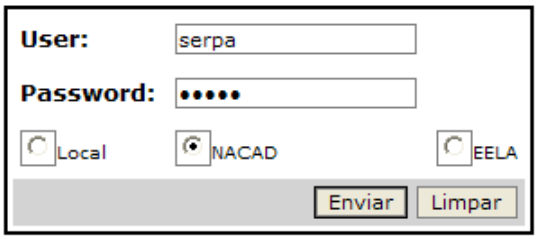


Figura 4.16: Interface de Login

O usuário deve informar sua identificação e senha, bem como selecionar em qual ambiente deseja submeter seus experimentos.

É importante ressaltar que conforme a escolha do ambiente pelo usuário, o menu principal apresentará diferentes opções de processamento.

Para esta demonstração, assume-se que o usuário esteja fazendo seu primeiro acesso ao ambiente, selecionou a opção "NACAD" no momento do *login* e realizará um experimento de categorização com o algoritmo bayesiano.

Após efetuado o *login*, apresenta-se o menu principal, mostrado na figura 4.17.

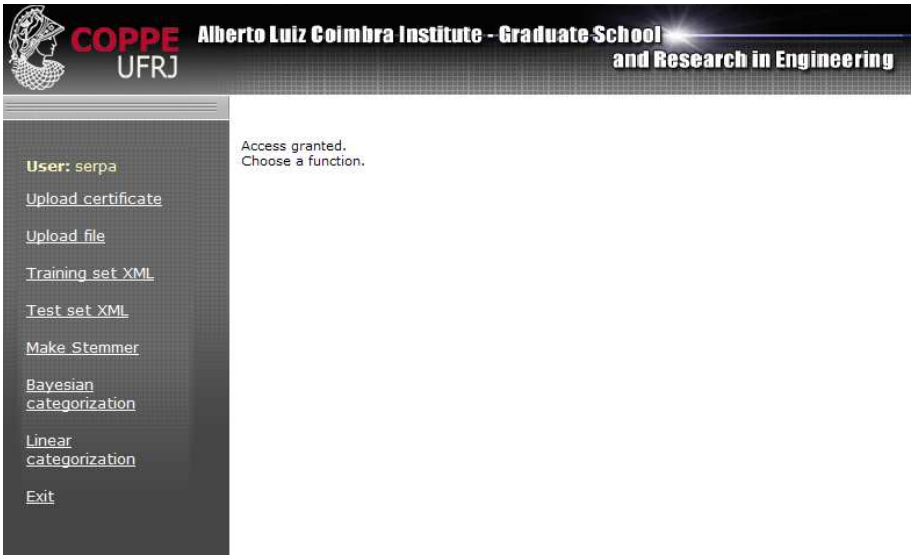


Figura 4.17: Menu Principal

Todo usuário ao se conectar no sistema, recebe uma área de processamento exclusiva na máquina do portal. Todos os arquivos, resultados e certificados são colocados nesta área.

Antes do usuário iniciar os experimentos é necessário que o mesmo faça o carregamento de seu certificado. Para isso o usuário deve utilizar o menu "*Upload certificate*", selecionar seu certificado e realizar o carregamento, conforme apresentado na figura 4.18.

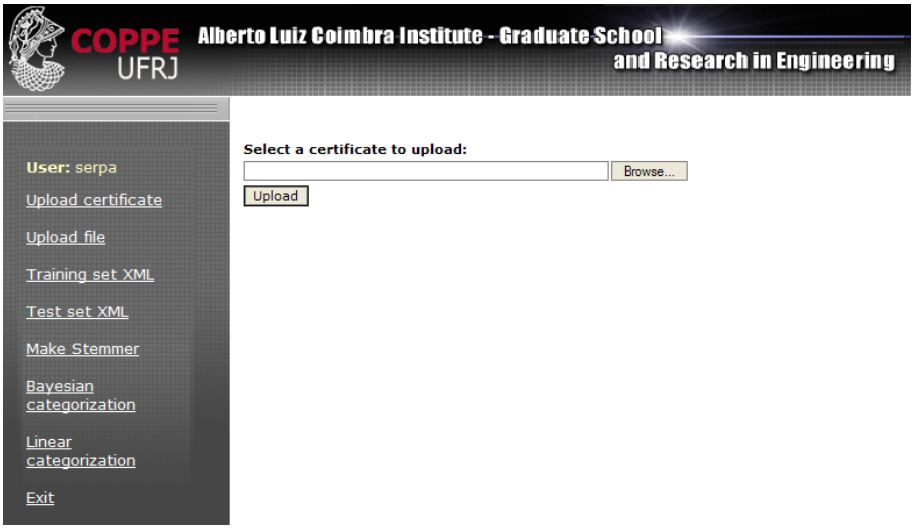


Figura 4.18: Carregamento do certificado

Todos os arquivos necessários para o processamento, seja do algoritmo bayesiano, seja do algoritmo de ranqueamento linear, devem estar no diretório do usuário na máquina do portal. Assim, deve ser realizado o carregamento desses arquivos em formato texto conforme ilustrado na figura 4.19. O usuário deve escolher o menu "*Upload file*", selecionar o arquivo e proceder o carregamento. Caso os arquivos estejam compactados, o sistema realiza a descompactação automaticamente.

Figura 4.19: Carregamento de arquivo comum

Após o carregamento dos arquivos, devem ser gerados os arquivos de treinamento e teste, que estão no formato "xml", que contém todos os textos selecionados para treinar e testar o categorizador. O usuário, também, pode realizar o carregamento destes arquivos, sem a necessidade de realizar os procedimentos de geração. Nas figuras 4.20 a 4.28 são ilustrados

os passos necessários para a geração dos arquivos de treinamento e teste. O arquivo de *stems* pode ser gerado da mesma maneira.

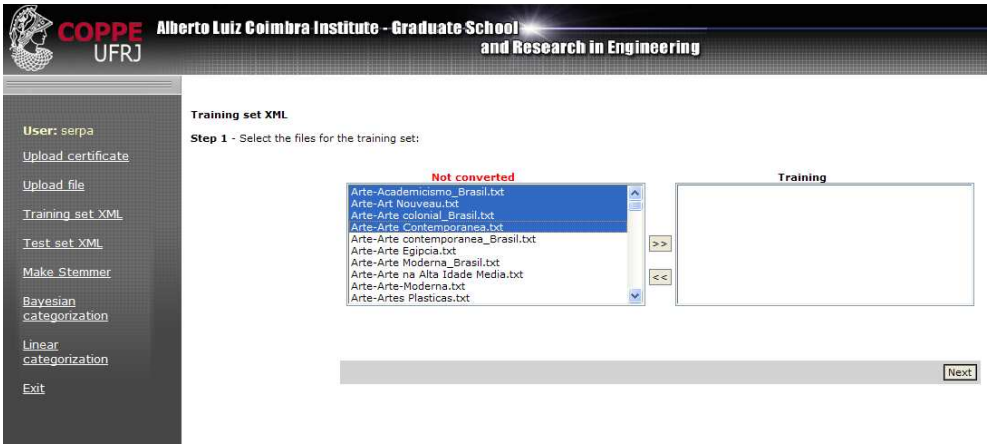


Figura 4.20: Geração do arquivo de treinamento - Passo 1. O usuário deve selecionar os arquivos que serão usados para treinar o modelo

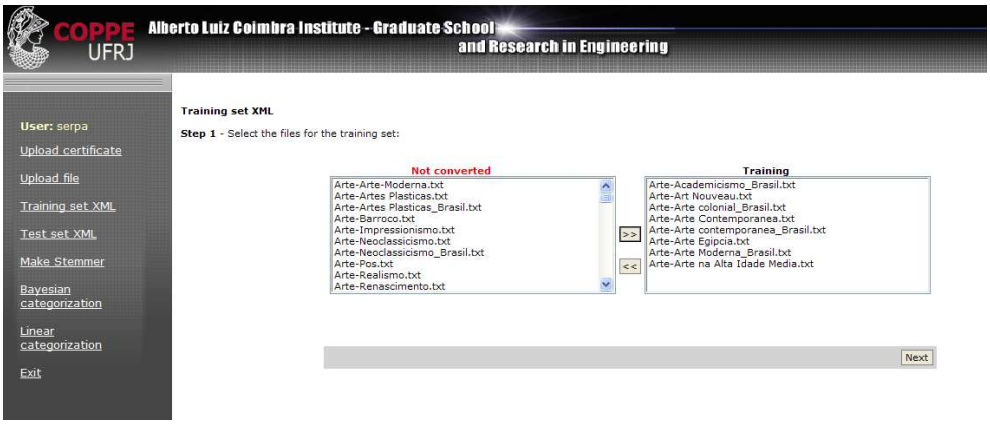


Figura 4.21: Geração do arquivo de treinamento - Passo 1. O usuário deve confirmar a seleção e selecionar a opção “Next”

Os textos que são coletados, antes de sua efetiva utilização pelo sistema, devem sofrer uma espécie de limpeza, com o objetivo de retirar caracteres de controle, excesso de espaços, sinais de pontuação etc, conforme a determinação do especialista. Vale lembrar que esta atividade ainda não é a retirada de *stopwords* e sim uma limpeza geral no texto, que consiste na análise léxica realizada na fase de pré-processamento.

A técnica de *case folding* foi implementada. Segundo Lopes [52], *case folding* consiste na conversão dos dados textuais para o mesmo tipo de caracter, ou seja, maiúsculo ou minúsculo. No presente trabalho o sistema converte todos os caracteres para minúsculo.

A limpeza consiste na verificação, caracter a caracter, de elementos indesejáveis nos textos, fazendo uma comparação com elementos de um vetor de caracteres previamente de-

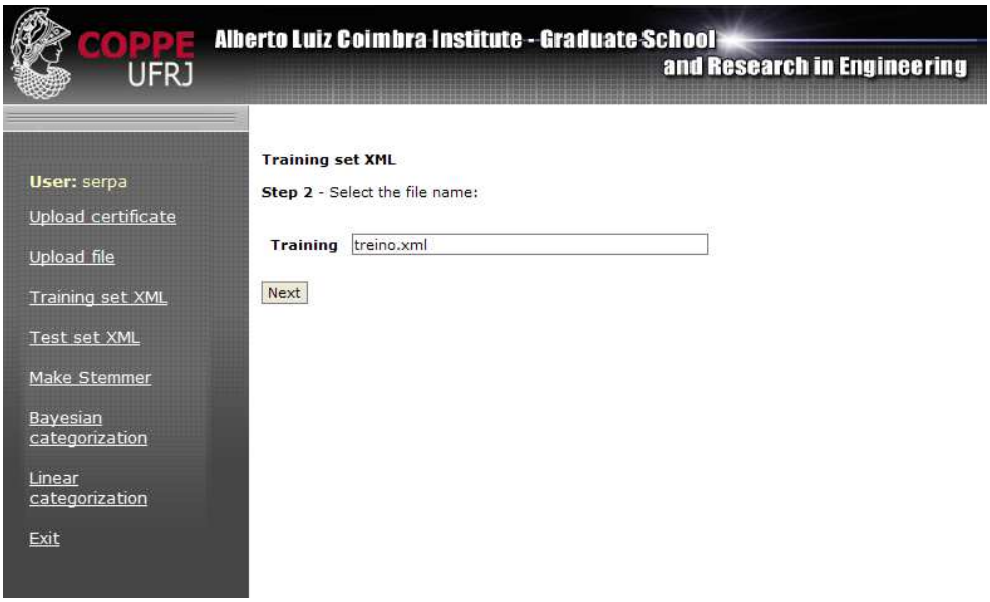


Figura 4.22: Geração do arquivo de treinamento - Passo 2. O usuário deve informar o nome do arquivo de treinamento e selecionar a opção “Next”

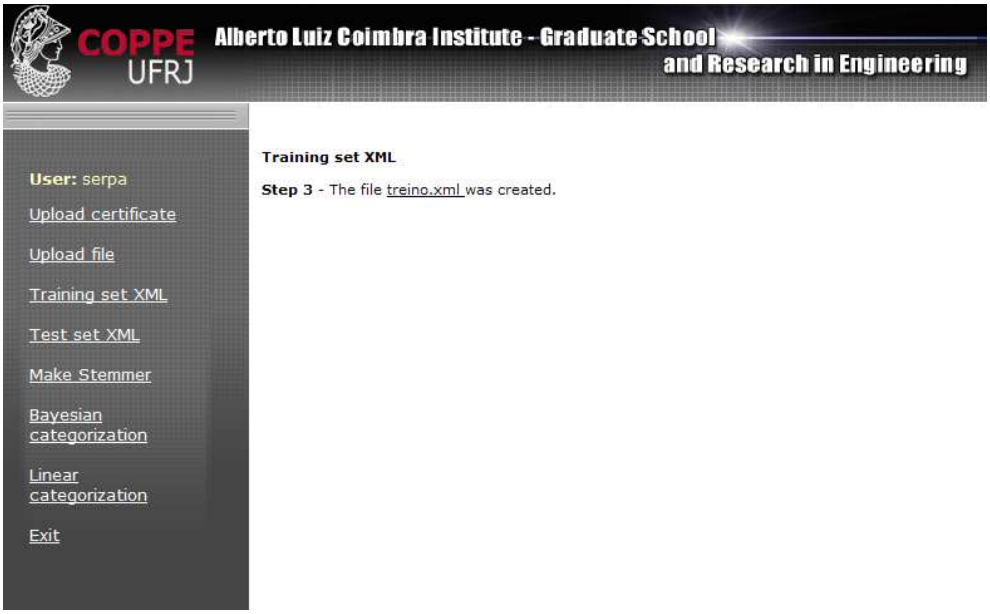


Figura 4.23: Geração do arquivo de treinamento - Passo 3. O arquivo de treinamento foi gerado e disponibilizado um link para *download* ou visualização

terminado. Para isso, foi utilizado um algoritmo que realiza uma pesquisa binária neste vetor.

A pesquisa ou busca binária[1] é um algoritmo de busca em vetores que requer acesso aleatório aos elementos do mesmo. Ela parte do pressuposto de que o vetor está ordenado e realiza sucessivas divisões do espaço de busca comparando o elemento buscado (chave) com o elemento no meio do vetor. Se o elemento do meio do vetor for a chave, a busca termina

[1]Na literatura em língua inglesa: *binary search algorithm*

com sucesso. Caso contrário, se o elemento do meio vier antes do elemento buscado, então a busca continua na metade posterior do vetor. E finalmente, se o elemento do meio vier depois da chave, a busca continua na metade anterior do vetor. A complexidade desse algoritmo é da ordem de $log_2 n$, onde n é o tamanho do vetor de busca.

Cada caracter é concatenado a uma saída no formato *string*. A soma de todas estas saídas é o texto final, livre dos caracteres não desejáveis. Esta verificação ocorre no momento da conversão dos textos para o formato *XML*, tanto na geração da base de treinamento como da base de teste, garantindo a integridade dos mesmos.

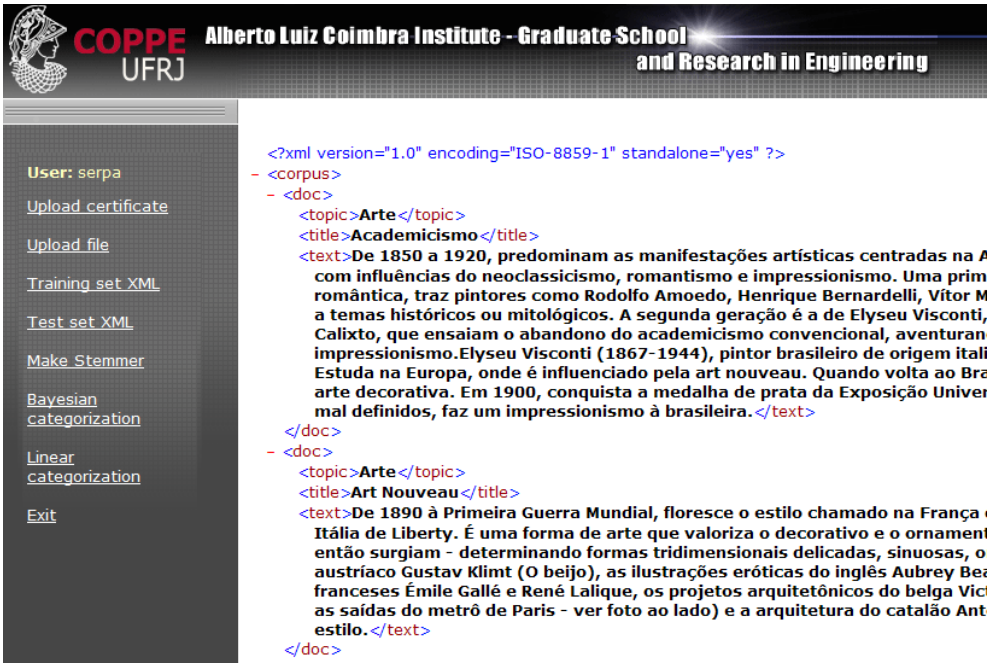


Figura 4.24: Visualização do arquivo de treinamento. Exemplo de um arquivo de treinamento sendo visualizado no *browser*

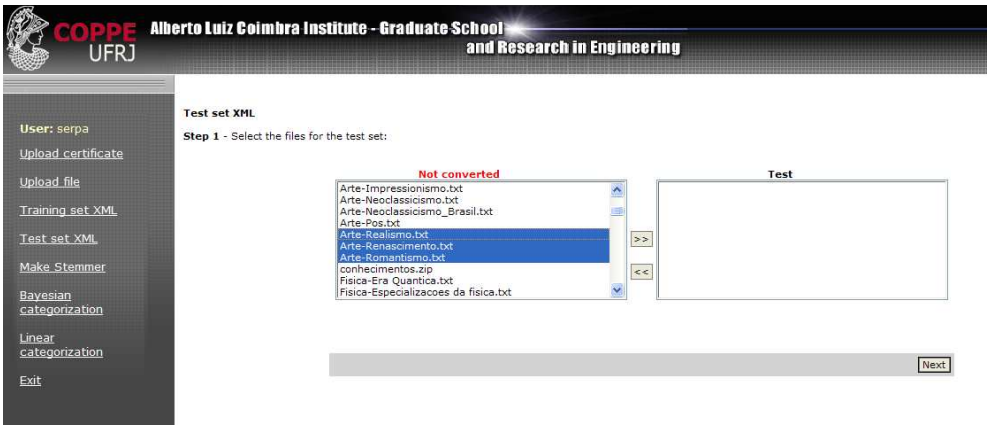


Figura 4.25: Geração do arquivo de teste - Passo 1

Após a geração dos arquivos de treinamento, teste e, opcionalmente, do arquivo de

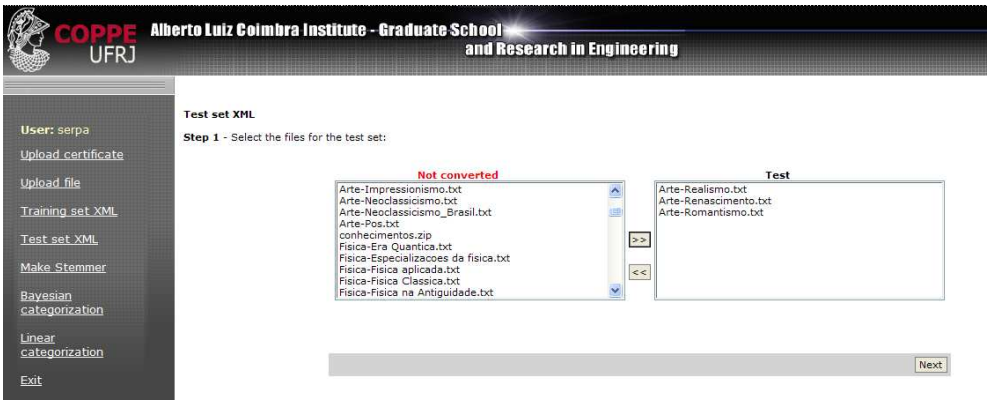


Figura 4.26: Geração do arquivo de teste - Passo 1

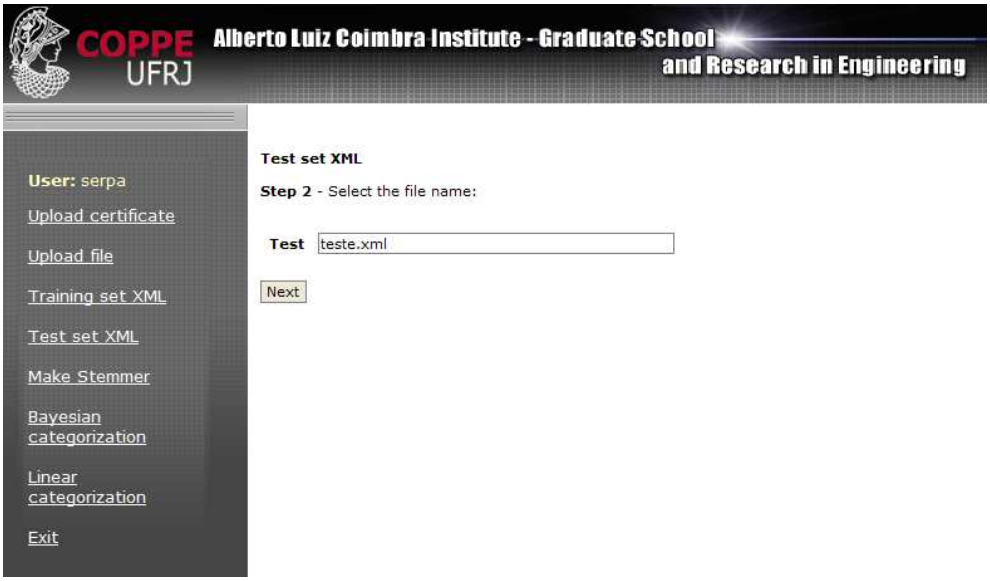


Figura 4.27: Geração do arquivo de teste - Passo 2

stems, é possível realizar a configuração das propriedades para a submissão do experimento. Na figura 4.29 é ilustrada a interface para a configuração das propriedades do algoritmo bayesiano.

Esta interface possibilita a inclusão de várias propriedades inerentes ao algoritmo em questão, como também em qual recurso disponível este será submetido. A lista dos recursos disponíveis é obtida no momento em que o usuário faz seu *login*. Na figura 4.30 é ilustrado um exemplo de configuração de propriedades.

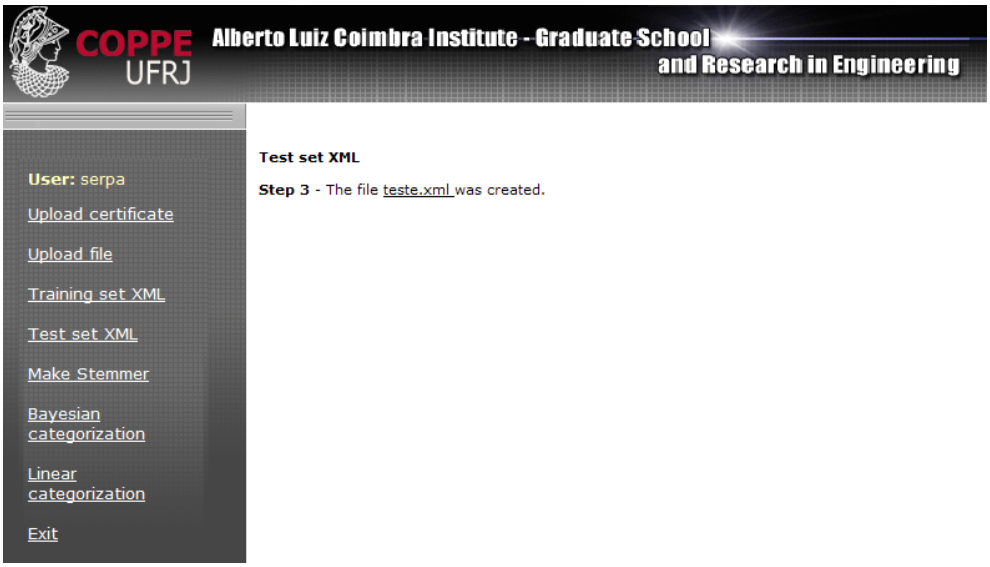


Figura 4.28: Geração do arquivo de teste - Passo 3

Para cada recurso selecionado, deve ser configurado seu respectivo conjunto de propriedades. Desta maneira, no momento da submissão, os elementos informados nas propriedades serão enviados para o recurso solicitado.

Após a submissão, alguns arquivos são gerados no recurso e copiados para a área do usuário no portal, para posterior *download*. Os arquivos são identificados com um nome composto por seu tipo (pos, neg e métrica) e o recurso em que foi submetido. O arquivo denominado "pos" contém os textos classificados positivamente, de acordo com a propriedade "*keyword*" informada. O arquivo "neg" contém o conjunto de textos classificados negativamente, também de acordo com a "*keyword*". Já o arquivo "métrica" contém as medidas "*precision*", "*recall*" e "*f-measure*" obtidas quando da execução do algoritmo. Na figura 4.31 é ilustrada a interface de retorno gerada.

As métricas de retorno, assim como os arquivos com as classificações podem ser baixados para posterior utilização. Algumas observações são tecidas na conclusão deste trabalho em relação à interface implementada e a possíveis modificações que poderiam ser realizadas para facilitar o uso do ambiente.

A grande facilidade proporcionada pelo portal Aîuri, consiste no encapsulamento das diversas atividades relativas à geração dos modelos bayesiano e de ranqueamento linear. Como já mencionado anteriormente, o sistema também possui funcionalidades da maior importância para as atividades de pré-processamento, como a geração dos arquivos contendo os textos de treinamento e teste e a implementação do arquivo com os *stems*, caso o usuário

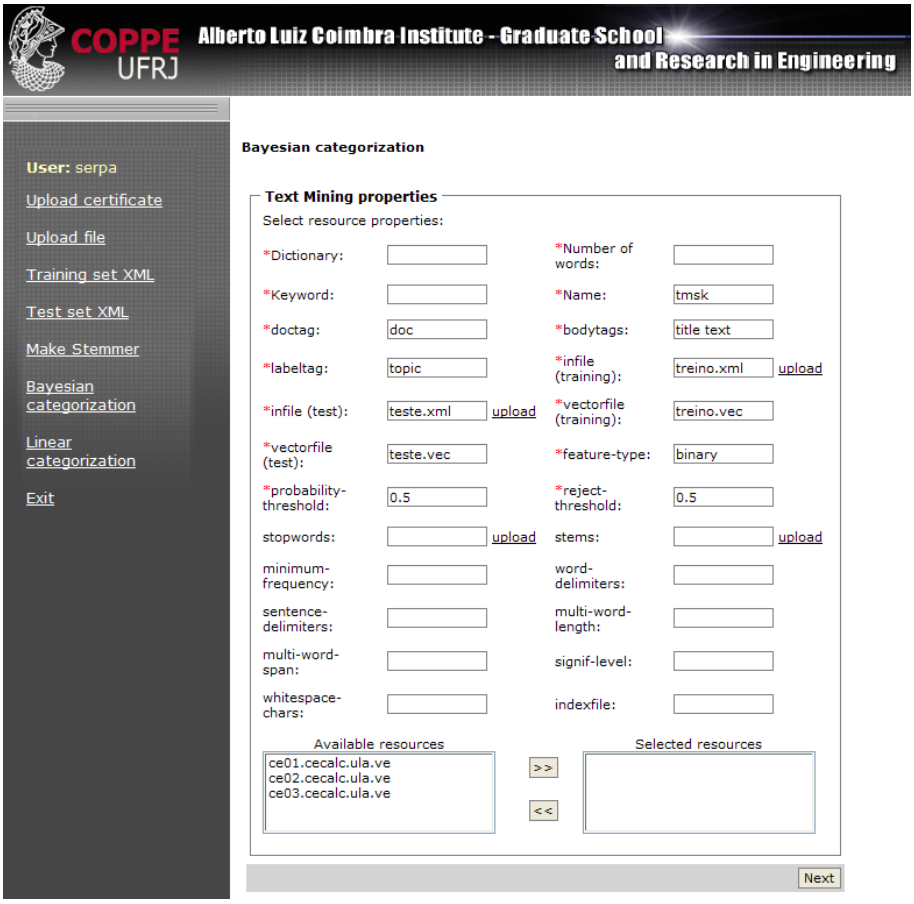


Figura 4.29: Interface de propriedades para o algoritmo bayesiano

julgue necessário.

Todavia, as funcionalidades implementadas no portal para as atividades de extração de padrões, mais especificamente, com a utilização das técnicas de categorização, possuem um diferencial, uma vez que encapsulam as tarefas necessárias à geração do modelo, de maneira que fiquem transparentes ao usuário. Na figura 4.32 são ilustradas as atividades encapsuladas pelo software.

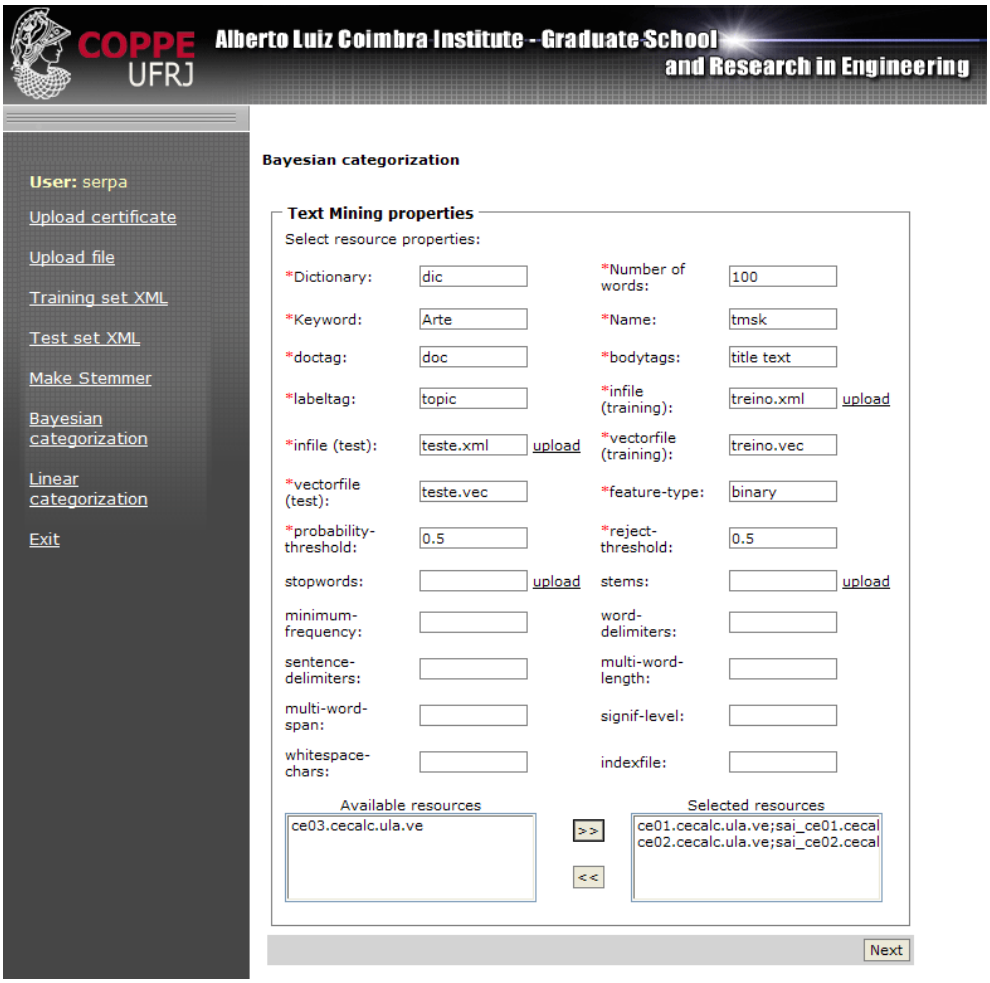


Figura 4.30: Propriedades configuradas para o algoritmo bayesiano

Após a realização das atividades de pré-processamento, o usuário deve realizar a configuração das propriedades do modelo selecionado (bayesiano ou linear). Esta configuração, como pode ser observado na figura 4.32, é realizada no *browser*. Uma vez que os parâmetros de execução estejam definidos, estes são enviados ao nó de execução selecionado, juntamente com os arquivos necessários, seguindo os padrões estabelecidos pelo ambiente de execução, ou seja, ou execução local, ou no *grid* NACAD, ou no *grid* EELA, que possui uma estrutura convencional.

Após a configuração das propriedades, devem ser gerados o dicionário, os vetores de treino e teste, um dos modelos selecionados e, finalmente, o teste do modelo, gerando os respectivos arquivos de saída, que são disponibilizados para o usuário.

Logo, uma contribuição evidente deste portal é a transparência com que atividades complexas são realizadas, de maneira a não delegar ao usuário final atividades que outrora estariam sob sua responsabilidade.

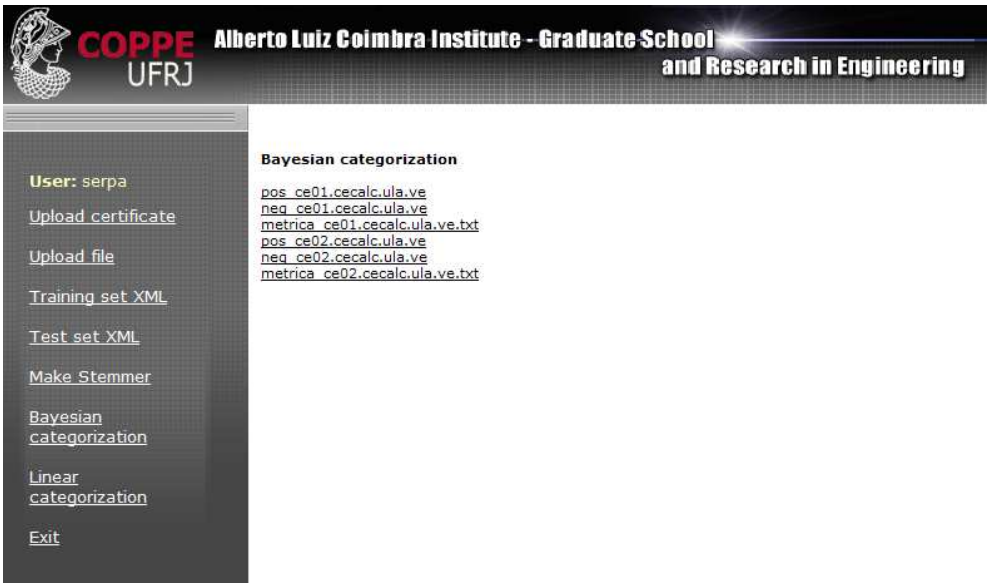


Figura 4.31: Retorno de submissão através do portal Aîuri

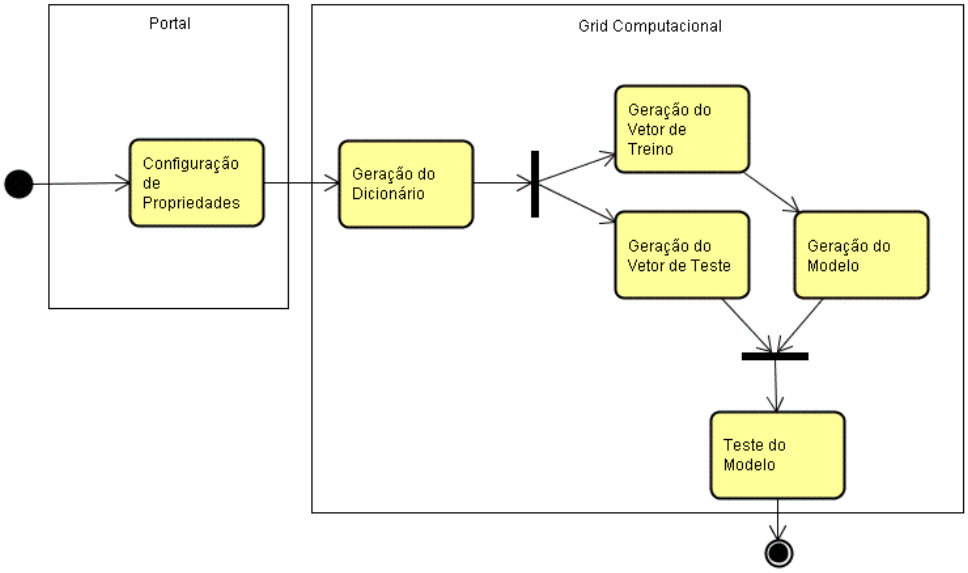


Figura 4.32: Atividades encapsuladas pelo Portal Aîuri

4.14 Atividades de Integração dos Ambientes de *Grid*

4.14.1 Integração no *Grid* GT4

O Projeto Aîuri surgiu de uma iniciativa do Núcleo de Transferência de Tecnologia (NTT) da COPPE/UFRJ, com o objetivo de se construir um sistema com interface *web* que viabilizasse a execução de experimentos na área de mineração de textos.

O sistema foi implementado no ano de 2006. Porém, para uma parte significativa dos experimentos na área de mineração de textos, o poder computacional é um fator primordial.

Diante deste cenário, foram realizadas alterações no sistema, de maneira a disponibilizar uma interface para um ambiente de *grid* computacional. O ambiente escolhido foi o suportado pelo *Globus Toolkit 4*. A seguir são descritas as principais atividades realizadas para a implementação desta interface.

4.14.1.1 O ambiente do GT4

O instalação do ambiente GT4 envolve uma série de ferramentas. Os procedimentos de instalação foram realizados utilizando as instruções disponíveis em [107]. O *grid* NACAD utiliza a versão Globus 4.0.4 possuindo atualmente cinco máquinas. Este ambiente utiliza o *PostgreSQL* e o Tomcat5.5 (como requisito de execução para o WebMDS)

O Globus também pode ser utilizado com outros sistemas gerenciadores de bancos de dados (SGBD) além do PostgreSQL, porém como ele foi desenvolvido e testado com este SGBD, optou-se pela utilização do PostgreSQL.

4.14.1.2 Criação das Classes Serviço *Web*

Para a integração do sistema Aîuri à infra-estrutura do GT4, foram realizadas alterações para que o sistema possa interagir com este novo ambiente. A principal e mais importante modificação, foi a criação da classe *WebServiceInterface*, que é uma interface para os serviços *web* que encapsulam as chamadas aos algoritmos implementados nas classes de negócio. Esta classe pode ser visualizada na figura 4.33.

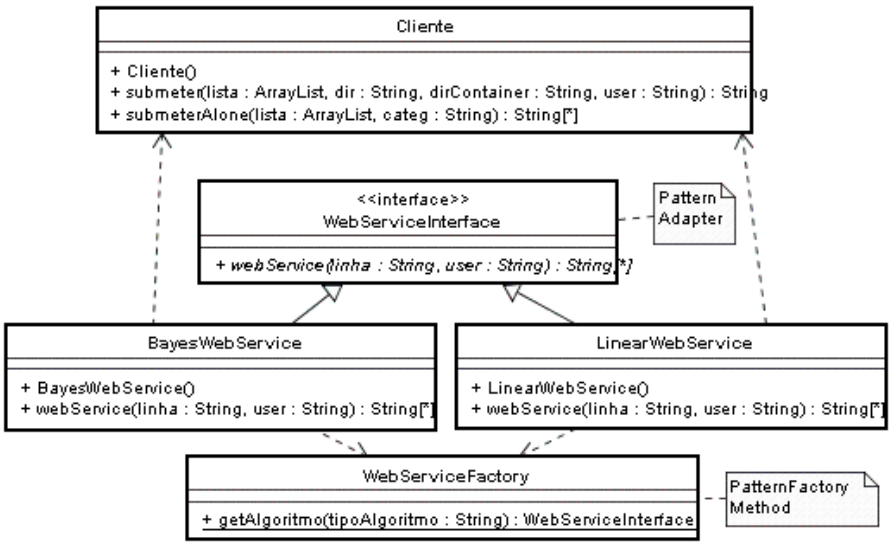


Figura 4.33: Modelo de classes implementado para a classe *WebServiceInterface*

Observa-se que o padrão de projeto *Adapter* é implementado nesta classe, que está definido na seção 4.5. Esta interface tem que ser herdada por qualquer classe em que algum algoritmo seja implementado e que necessite ser chamado via serviço *web*. O método *webService()*, que é implementado através da classe filha, encapsula todas as chamadas aos métodos das classes de negócio que, de fato, possuem a implementação dos algoritmos. Para a criação de instâncias destas classes é utilizado o padrão de projeto *Factory Method*, também definido na seção 4.5.

As atividades encapsuladas estão apresentadas nas figuras 4.32 e 4.34.

```
1.  ...
2.  // configurar dicionário
3.  DicionarioModelo dicionario = new DicionarioModelo();
4.  ...
5.  if (dicionario.gerarDicionario(p, user)) {
6.      ...
7.      // gerar vetor de treino
8.      ...
9.      // gerar vetor de teste
10.     ...
11.     BayesModelo b = (BayesModelo) AlgoritmoFactory.getAlgoritmo("bayes");
12.     bayesDTO = b.gerarEstatisticas(p, sai, pos, neg, user); // encapsula
13.     // a geração e o teste do modelo de classificação
14.     String metricas = b.salvarMetricas(bayesDTO, time, maquina, user);
15.     ...
16. }
```

Figura 4.34: Trecho de código de um serviço *web* para o classificador *Naive Bayes*

A chamada ao serviço *web* é feita pelo método *submeter()* da classe "Cliente", que é invocado quando o usuário solicita a submissão de seu experimento através do *browser*. Na figura 4.33 é ilustrada a classe "Cliente". O método *submeter()* é o responsável por transferir para o nó do *grid*, os arquivos e os parâmetros necessários para o algoritmo que será executado. Na figura 4.35 é ilustrado um trecho do código de submissão.

4.14.1.3 Publicação do Serviço *Web*

O GT4 faz extenso uso de mecanismos de serviços *web* para definir suas interfaces e as estruturas de seus componentes. Serviços *web* provêem flexibilidade, extensibilidade e adotam mecanismos baseados em XML para descrever, descobrir e invocar serviços de rede.

```
1.  ...
2.  String recurso = "http://"+ maquina + ":8086/Aiuri2/wsdl/BayesModelo.wsdl";
3.  ...
4.  ///* para uso com o globus4
5.  URL url = new URL(recurso);
6.  Client c = new Client(url);
7.  // verificar o que precisa ser copiado (stemm, stoplist) testar para cada argumento
8.  copy = new Copy();
9.  if (!p.getInfile().equals()) {
10.     fromURL = "gsiftp://"+ Constants.LOCALHOST + "/"+ p.getInfile();
11.     toURL = "gsiftp://"+ maquina + "/"+ p.getInfile();
12.     copy.copyFile(fromURL, toURL); // copia o arquivo para o nó de execução
13. }
14. ...
15. String webService = new String("wsBayes");
16. // invoca o serviço web passando os parâmetros necessários
17. resultado = c.invoke(webService, new Object[] new String(linha), new String(dir));
18. // copiar de volta o arquivo dos classificados positivamente
19. toURL = "gsiftp://"+ Constants.LOCALHOST + "/"+ pos;
20. fromURL = "gsiftp://"+ maquina + "/"+ pos;
21. copy.copyFile(fromURL, toURL);
22. ...
```

Figura 4.35: Trecho de código do método submeter copiando os arquivos necessários para a execução do serviço web

O sistema Aîuri, no módulo desenvolvido para o GT4, adere aos princípios desta abordagem. Os procedimentos para geração do serviço *web* podem ser realizados em qualquer máquina que possua o Java 1.4 ou superior, o servidor de aplicações Tomcat5.5 ou superior e a ferramenta IDE Eclipse3.2.

Inicialmente, deve-se abrir o código-fonte no Eclipse e selecionar a classe para a qual será gerado o serviço *web*. Este procedimento é ilustrado na figura 4.36.

Em seguida deve-se realizar os procedimentos para a criação de um serviço *web* descritos pela ferramenta. Na figura 4.37 é ilustrada a interface para este procedimento. As opções *default* devem ser mantidas.

A interface ilustrada na figura 4.38 permite a seleção dos métodos da classe que serão publicados como serviço *web*. Após esta etapa, o serviço *web* é criado e é solicitada a inicialização do Tomcat. Após sua inicialização o serviço *web* está criado e o procedimento pode ser encerrado.

Um arquivo com o mesmo nome da classe que originou o serviço *web* é criado com a

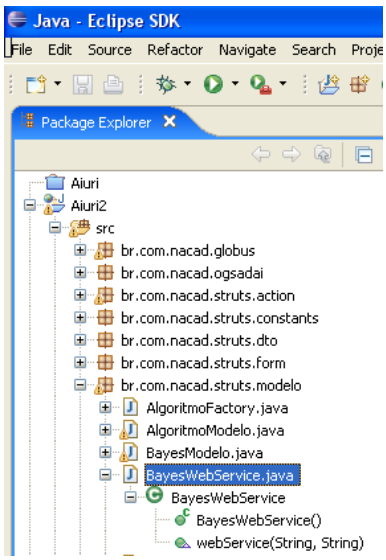


Figura 4.36: Seleção de uma classe para a geração de um serviço *web*

extensão "wsdl". Na figura 4.39 é ilustrado o arquivo gerado.

O projeto deve ser exportado como um arquivo "war" para ser instalado na máquina do *grid*. Este arquivo deve ser copiado para o diretório *webapps* do Tomcat. Após a inicialização do Tomcat é criado um diretório com o nome do arquivo "war". Deve-se, então, editar o arquivo "wsdl" e incluir na *tag address location*, o nome ou o IP da máquina do *grid*. Na figura 4.40 é ilustrada a *tag* a ser modificada. Quando o *container* for reiniciado, o serviço *web* será publicado automaticamente.

4.14.1.4 Publicação do Serviço *Grid*

O sistema Aîuri permite a utilização de duas abordagens para serviços *web*. A primeira é a utilização do serviço *web*, publicado no *container* Tomcat ou outro servidor de aplicações. A outra abordagem consiste na publicação do serviço como um serviço *grid*. Este serviço é publicado no *container* do GT4.

Na versão atual do sistema Aîuri, está implementada com a abordagem discutida na seção 4.14.1.3, porém a publicação de um serviço *web* como um serviço *grid* envolve pouco esforço de desenvolvimento.

Os procedimentos para a criação e publicação de um serviço *grid* são muito similares aos de um serviço *web*, contudo, envolve alguns detalhes que são descritos a seguir:

(i) Definição da interface do serviço: isto é realizado com a criação do arquivo *WSDL*, porém deve-se adicionar as *tags wsdlpp* e *wsrp*;

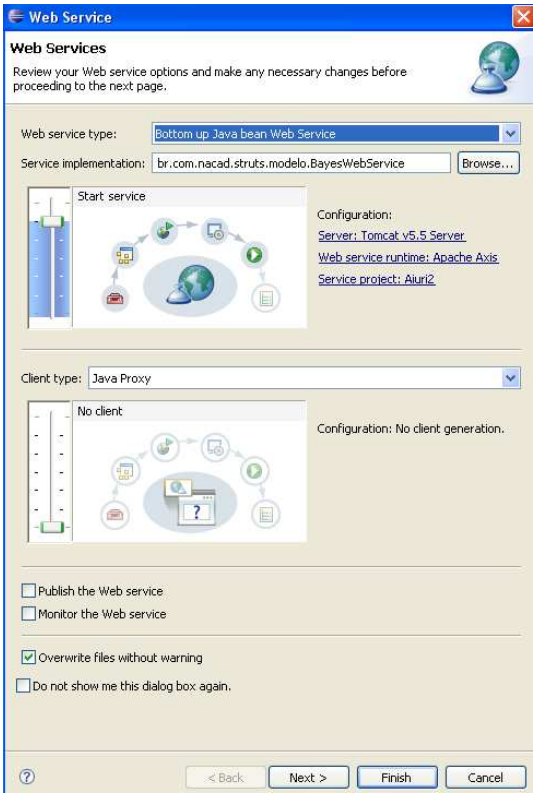


Figura 4.37: Criação de um serviço *web*

(ii) Implementação do serviço *web*: o sistema Aîuri implementa os serviços *web* BayesWebService e LinearWebService. Nestes serviços devem ser implementados o método *public ResourcePropertySet getResourcePropertySet()*. Deve-se ressaltar que a implementação de serviços *web* é independente da linguagem de programação.

(iii) Definição de parâmetros para a publicação do serviço: os parâmetros são definidos nos arquivos *WSDD* e *JNDI*.

(iv) Compilação e geração do arquivo *GAR*: o projeto deve ser compilado e encapsulado em um arquivo com extensão *GAR*, para então ser publicado no *container* do GT4. Na figura 4.41 são ilustrados todos os procedimentos descritos.

Mais detalhes sobre a implementação, ver [90].

4.14.2 Integração no *Grid* EELA

O projeto EELA, como uma de suas atribuições, tem o objetivo de disponibilizar ambientes para que aplicações diversas possam ser portadas para sua infra-estrutura. A escola de *grids* ou EGRIS (*EELA Grid School* - Escola de *Grid* EELA) realiza esta tarefa, tornando os

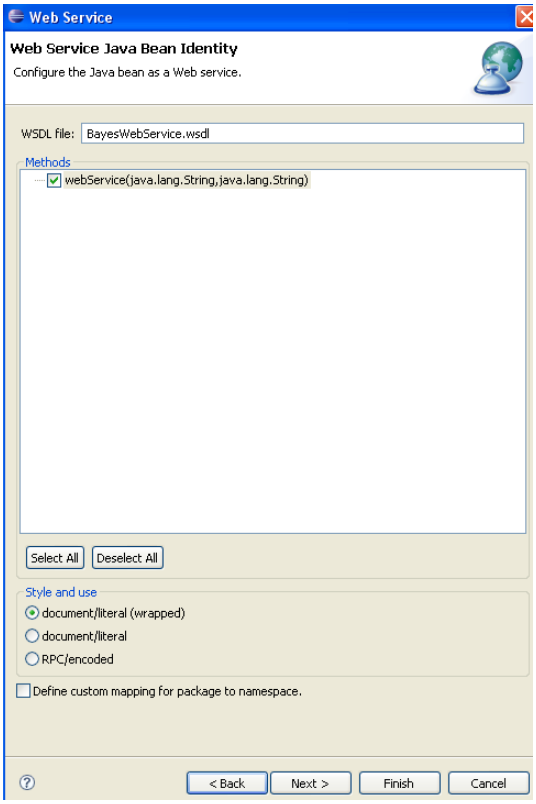


Figura 4.38: Seleção dos métodos da classe

Figura 4.39: Arquivo “wsdl” criado

seus participantes aptos a “gridificar” suas aplicações, isto é, migrá-las para um ambiente de *grid* real, com a disponibilização de toda a infra-estrutura necessária.

Neste contexto, o Projeto Aîuri foi selecionado para participar da EGRIS-2 (Segunda Escola de *Grid* EELA), realizada em Mérida, na Venezuela, em agosto de 2007.

Nesta escola, houve a possibilidade de testar o sistema em um *grid* real. Para isso, foram realizadas alterações, que possibilitaram a implementação de uma interface que viabilizou seu uso para este novo ambiente.

A seguir são descritas as principais tarefas realizadas para a implementação da interface para o *grid* EELA.

```
<wsdl:service name="BayesWebServiceService">
   <wsdl:port binding="impl:BayesWebServiceSoapBinding" name="BayesWebService">
      <wsdlsoap:address location="http://localhost:8080/Aiuri2/services/BayesWebService"/>
   </wsdl:port>
</wsdl:service>
```

Figura 4.40: Trecho do arquivo “wsdl” com a *tag* a ser modificada

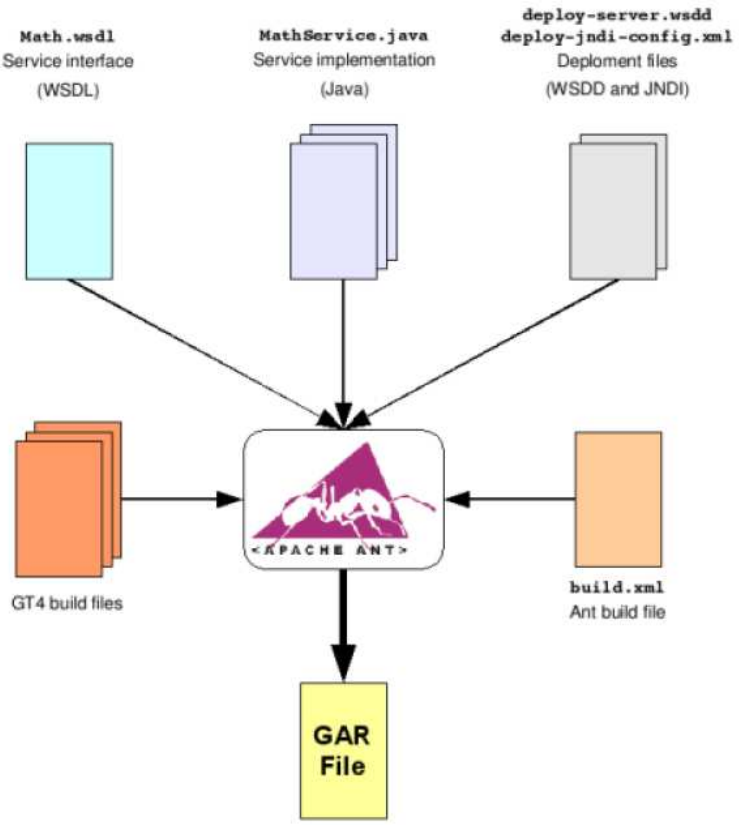


Figura 4.41: Atividades para a publicação de um serviço *grid*

4.14.2.1 Utilização do AMGA

AMGA [108] é um serviço de metadados para *grids* computacionais. Pode ser entendido como um serviço de acesso a banco de dados para aplicações em *grid*, que permitem que tarefas em execução no *grid* acessem bancos de dados, provendo autenticação, como também uma camada que esconde do usuário as diferenças entre os diversos bancos de dados, de maneira a tornar uniforme o acesso aos mesmos. De fato, o AMGA é um serviço que atua entre o SGBD e a aplicação cliente. Nas figuras 4.42 e 4.43 são ilustradas a estrutura e um esquema de acesso ao AMGA, utilizado pelo portal Aîuri.

A camada de apresentação de dados (*Data Presentation Layer*) consiste em todas as páginas *web* que habilitam os usuários a acessar os funcionalidades do AMGA. Na camada de lógica de aplicação (*Logic Application*) é encapsulada a implementação das funcionalida-

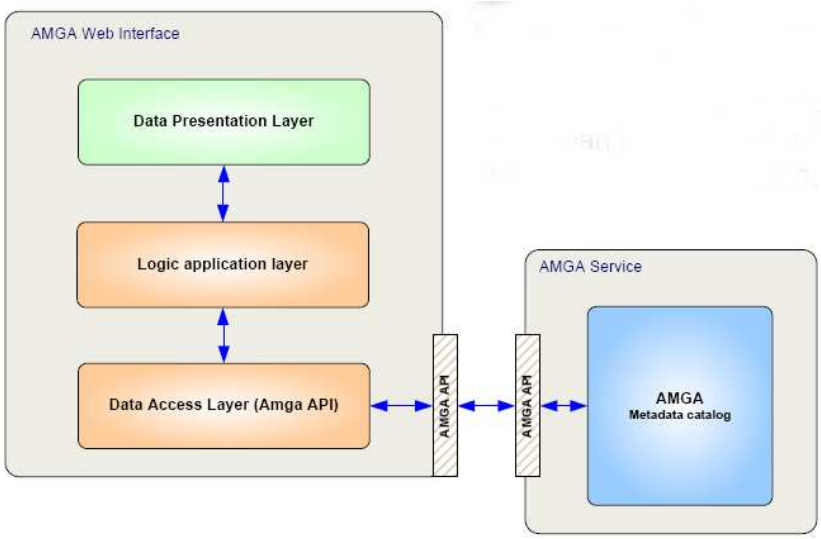


Figura 4.42: Exemplo esquemático da estrutura do AMGA

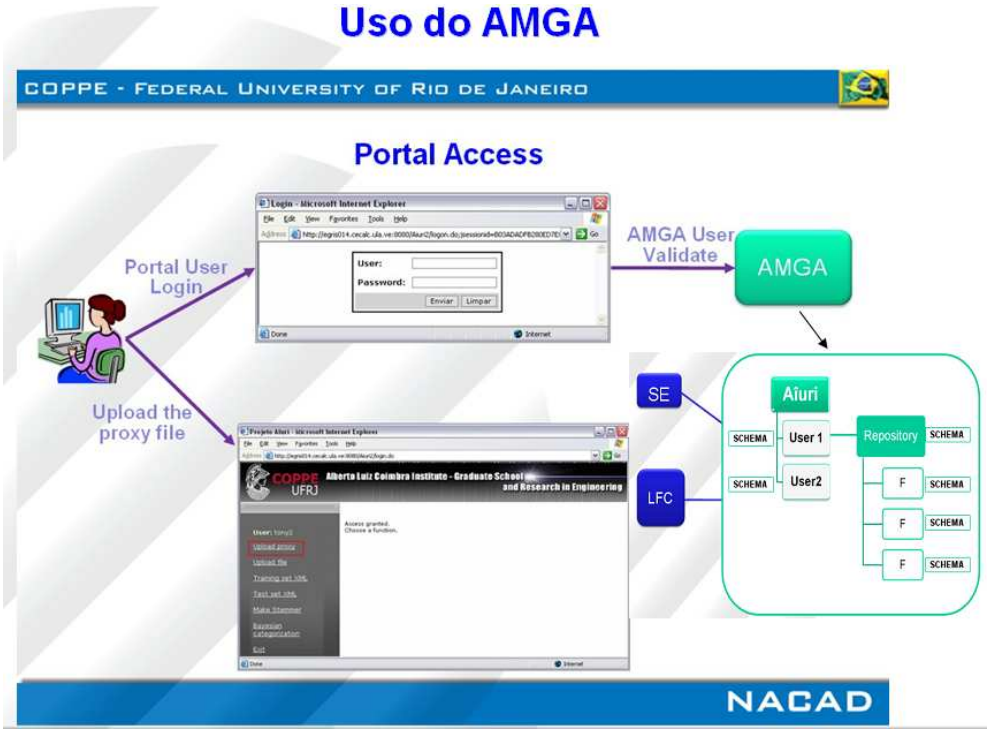


Figura 4.43: Exemplo de acesso ao portal Aîuri utilizando a estrutura do AMGA

des. Já a camada de acesso a dados, implementa os componentes de *software* que garantem a inclusão e a extração de dados no servidor AMGA.

No sistema Aîuri, o AMGA é utilizado para a criação de uma estrutura na qual o usuário é validado no momento do *login*, conforme ilustrado na figura 4.43.

A seguir são apresentadas as atividades que foram implementadas no sistema Aîuri para incluir o serviço AMGA:

(i) Criação dos atributos *amgaConfHost*, *amgaConfPort*, *amgaConfUseSSL*, *amgaConfRoot* e *defaultSE* na classe *Configurations*. Esta classe implementa o padrão de projeto *Singleton*, que foi já discutido na seção 4.5. Na figura 4.44 é ilustrada a classe *Configurations*;

(ii) Implementação na classe *UsuarioModelo* do método *validarLogin()*: neste método é criada a conexão com o servidor AMGA, habilitando o usuário a utilizar o sistema;

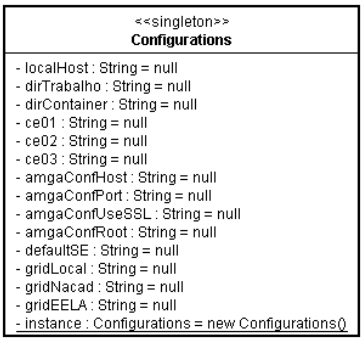


Figura 4.44: Classe "Configurations" que implementa o padrão de projeto *Singleton*

As atividades de criação dos metadados são realizadas através de uma interface *web* disponibilizada pelo projeto AMGA. Na figura 4.45 é ilustrada esta interface e, na figura 4.46 é ilustrada a estrutura criada para o portal Aîuri na EGRIS.

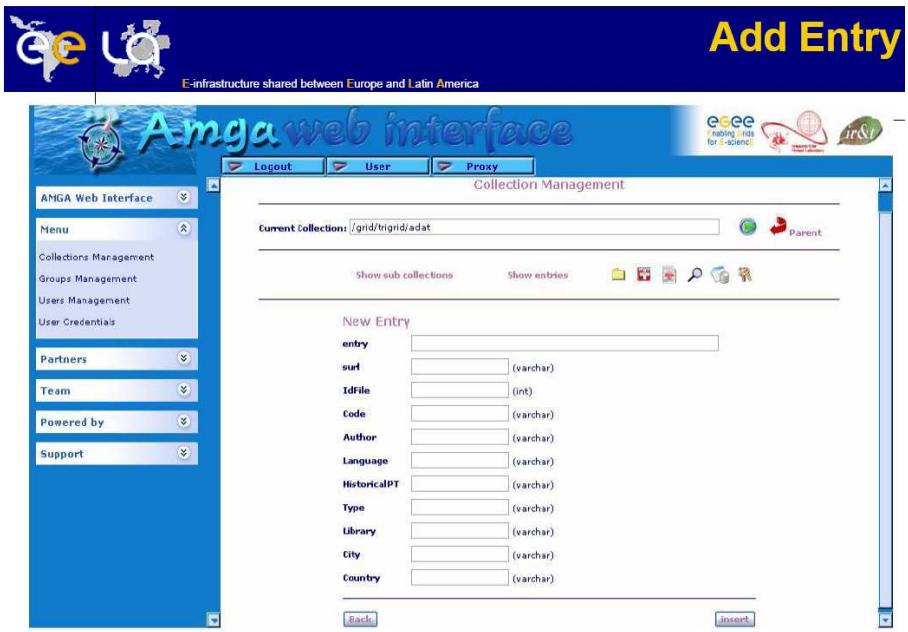


Figura 4.45: Interface *web* do AMGA

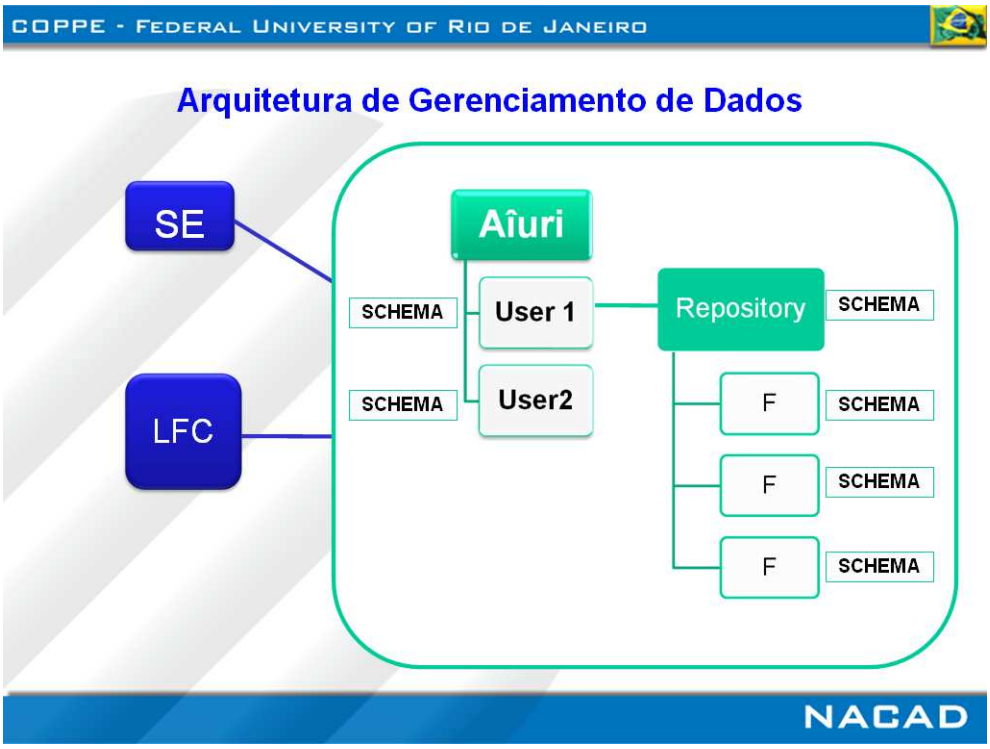


Figura 4.46: Estrutura criada no AMGA para o portal Aîuri

4.14.2.2 Utilização do GSAF

Nos ambientes de *grids* computacionais, arquivos são as menores unidades de dados, os quais devem ser armazenados em entidades denominadas *Storage Element* (Elemento de Armazenamento) ou SE. Os arquivos podem ser replicados em vários SE, por questões de ubiquidade, segurança ou necessidades de compartilhamento.

Relações entre a localização dos arquivos e réplicas, e seus identificadores, são mantidas com um serviço de catálogo de arquivos e, para cada arquivo é possível associar metadados descritivos organizados por meio de um serviço de catálogo de metadados específico. Na figura 4.47 é ilustrado um resumo dos três principais serviços de gerenciamento de dados que aplicações de *grid* necessitam usar, e as APIs relacionadas.

O GSAF (*Grid Storage Access Framework*) [109] é projetado de maneira que cada serviço é provido de uma *API* para os desenvolvedores e ferramentas de linha de comando para os usuários finais. Cada serviço é independente do outro e trabalham de modo "*stand alone*". Os componentes das aplicações devem referenciar os serviços de maneira desacoplada. Esta fragmentação força os desenvolvedores e os *web designers* a utilizarem uma arquitetura vertical para acessar qualquer serviço único e isso sempre conduz ao mesmo problema: a aplicação deve cuidar da coerência e da sincronização dos dados.

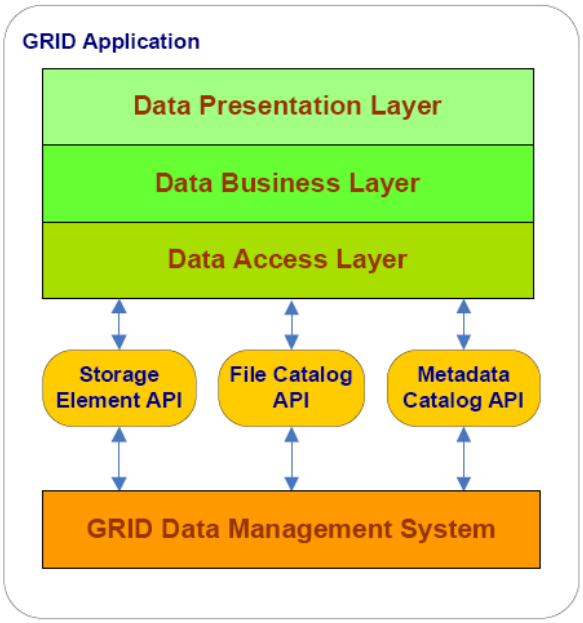


Figura 4.47: Serviços de gerenciamento de dados para *grid*

Da perspectiva do usuário final, os fatos são semelhantes. O usuário é capaz, usando ferramentas de linha de comando, de gerenciar objetos através do catálogo de arquivos em vez de usar o catálogo de metadados ou o SE, porém, toda a lógica de relacionamentos deve ser de seu conhecimento. Além disso, o usuário fica dependente da utilização de ferramentas instaladas em máquinas específicas, chamadas *User Interface* (Interface do Usuário) ou UI, localizadas no *grid*.

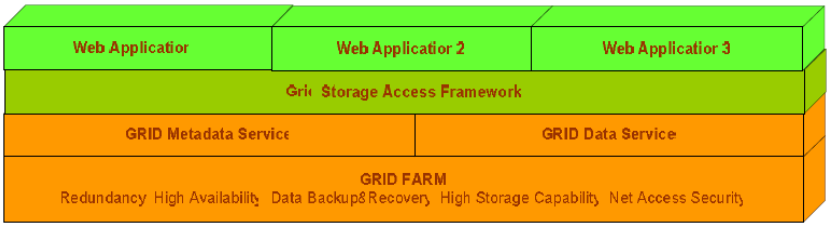
O GSAF tem o propósito de atender a estas necessidades de duas formas:

(i) Oferecendo uma ferramenta que permita aos desenvolvedores construirem aplicações, escondendo a complexidade e a fragmentação das várias APIs subjacentes, assegurando coerência e flexibilidade;

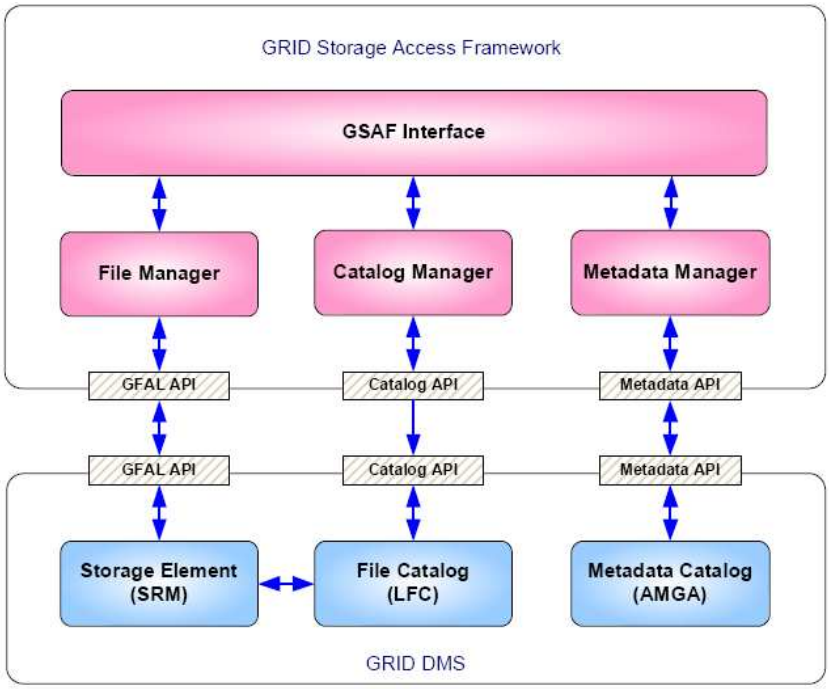
(ii) Prover uma interface de fácil utilização pelo usuário, para acessar sistemas de gerenciamento de dados remotamente via *web*.

Na figura 4.48 são ilustradas duas visões da arquitetura do GSAF, apresentando uma estrutura de camadas e os elementos de conexão entre elas.

O GSAF foi adicionado ao sistema Aîuri para realizar as atividades de catalogação de arquivos e gerenciamento de tarefas submetidas no *grid*. Porém, durante os testes de submissão de *jobs*, ocorreram problemas de compatibilidade das APIs do GSAF com o ambiente de desenvolvimento. Os problemas não foram solucionados a tempo durante a EGRIS, de



(a) Arquitetura em camadas do GSAF



(b) Exemplo esquemático da arquitetura do GSAF

Figura 4.48: Duas visões da arquitetura do GSAF

maneira que os testes com este importante *framework* não foram concluídos no ambiente de "gridificação". Na figura 4.49 é ilustrado um exemplo esquemático do que foi implementado no portal Aîuri utilizando o GSAF.

4.14.2.3 Submissão de jobs

Para a submissão de jobs no *grid* EELA é utilizada a API LCG (LHC Computing Grid) que disponibiliza várias funcionalidades para acesso a este ambiente.

Para a integração desta API no sistema Aîuri, foi criada a classe *Submit*, na qual são implementados os métodos de acesso ao *grid*. Na figura 4.50 é ilustrada a classe *Submit*.

Para a submissão de *jobs* utilizando esta API, é necessário um arquivo que descreva as propriedades das tarefas que serão executadas. A *Job Description Language* ou JDL [110], é a linguagem utilizada para realizar esta descrição. Na figura 4.51 é ilustrado um exemplo

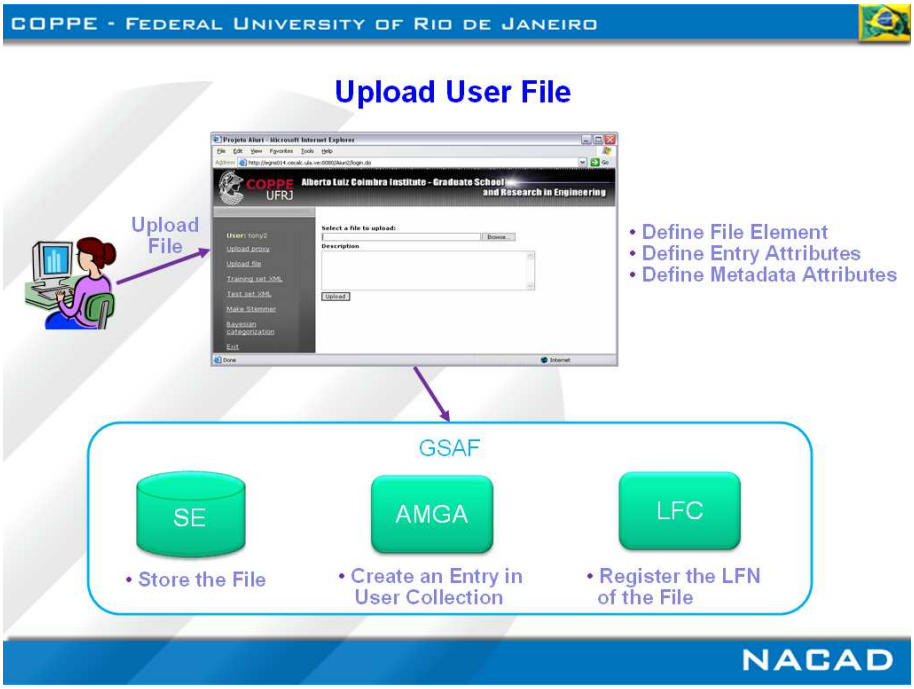


Figura 4.49: Exemplo esquemático de implementação do GSAF no portal Aîuri

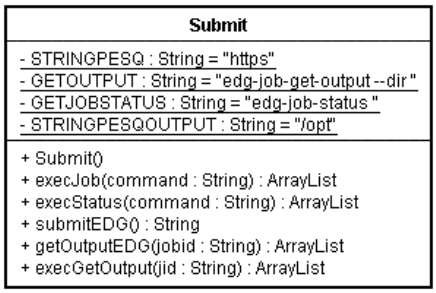


Figura 4.50: Classe *Submit*

de arquivo JDL.

Outros dois arquivos devem ser gerados para que se proceda a execução de um *job* no *grid*. São eles:

(i) *File submitter*: este é o arquivo responsável pela submissão do *job* que está sendo enviado ao *computing element*. É um *script .sh*, testado com o *shell Bash* do *Linux*, que é executado a partir do portal. O arquivo invocado pelo *submitter* está no formato JDL;

(ii) *File sh*: este arquivo é um *shell script* do *Linux*, que contém a seqüência de chamadas às classes Java que são executadas durante a execução do *job*. As variáveis de ambiente necessárias para que comandos Java possam ser executados também são definidas neste arquivo.

```
1.  Type="Job";
2.  JobType="Normal";
3.  Executable="/bin/hostname";
4.  Arguments="";
5.  StdOutput="stout.txt";
6.  StdError="stderr.txt";
7.  OutputSandbox={"stderr.txt", "stdout.txt"};
```

Figura 4.51: Exemplo de um arquivo JDL que descreve um *job* que retorna o *hostname* do nó de execução

Na figura 4.52 é ilustrada a seqüência de execução no ambiente EELA.

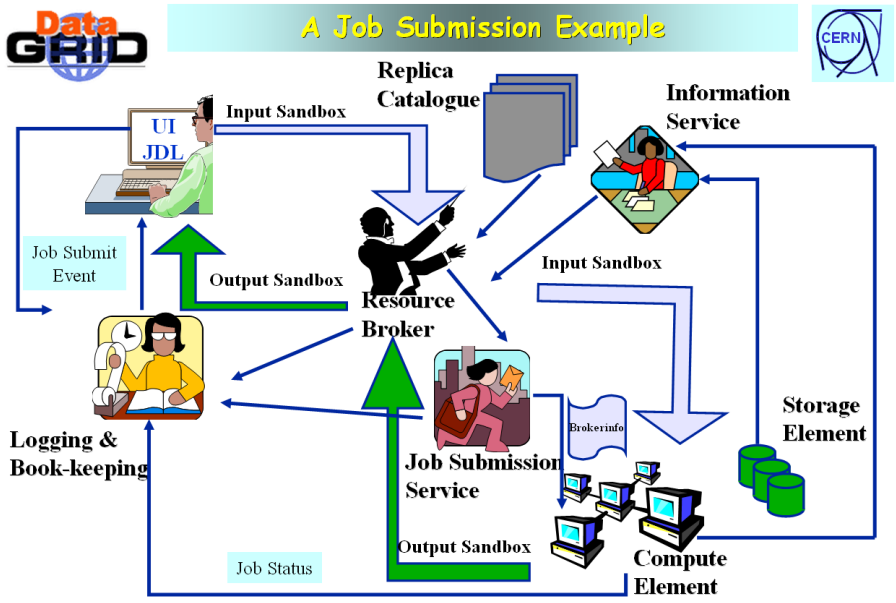


Figura 4.52: Seqüência de execução de um *job* no *grid* EELA

Todos os testes realizados com esta API obtiveram sucesso, permitindo ao sistema Aîuri ser integrado a mais um ambiente computacional. Na figura 4.53 é ilustrada uma interface do sistema Aîuri durante a execução de um *job* no *grid* EELA.

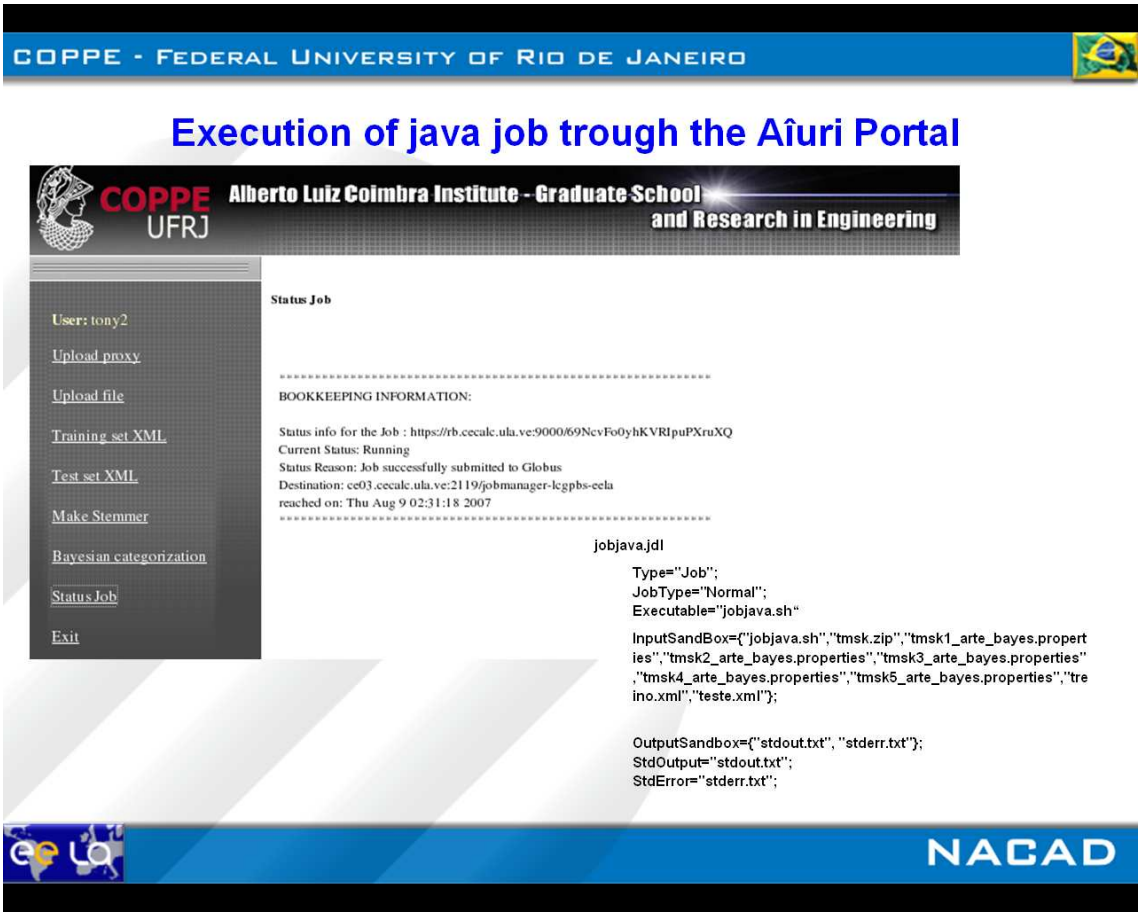


Figura 4.53: Exemplo do sistema Aîuri realizando a execução de um *job* no *grid* EELA

Capítulo 5

Testes do Portal Aîuri

A seguir são apresentados dois testes realizados para a avaliação da qualidade dos resultados gerados e validação dos ambientes pelo sistema Aîuri. São usados dois conjuntos de textos que já foram previamente classificados. Porém, esta classificação prévia é utilizada para a geração do modelo de categorização e, posteriormente, para a verificação dos textos categorizados.

São apresentados gráficos que ilustram os resultados dos modelos, com suas respectivas *f-measure* e os tempos de processamento para os melhores modelos gerados em um computador local e no *grid* NACAD.

Os testes foram realizados em dois ambientes distintos, sendo o primeiro ambiente uma máquina executando o portal Aîuri e o segundo o *grid* NACAD. A máquina em que é executado o portal Aîuri serve como ambiente de execução *stand alone*, porém sua principal função é servir como interface de submissão para os algoritmos que são executados no *grid*.

O objetivo principal do teste no *grid* foi a validação do ambiente como plataforma de execução de algoritmos para mineração de textos. Por isso, somente uma máquina foi utilizada, de maneira a emular um ambiente de *grid* computacional.

Desta maneira, entende-se que a escalabilidade do *grid* está mantida, onde a inserção de novas máquinas poderá ocorrer de maneira transparente, uma vez que o ambiente está implantado, testado e documentado. Também, deve-se ressaltar que o portal identifica novas máquinas integradas ao *grid*, de maneira que o usuário é capaz de visualizá-las no momento da definição dos parâmetros para seus experimentos, como está ilustrado na figura 4.29.

Outro importante fator a ressaltar, é que os algoritmos implementados no portal são se-

qüenciais, podendo ser executados em somente um nó do *grid*, comprovando que a execução em diferentes ambientes não afeta o desempenho dos algoritmos, conforme as medidas obtidas e que são comentadas a seguir.

5.1 Textos Utilizados

Para os estudos de caso foi utilizado o conjunto de textos CETENFolha [111], que é um *corpus* de cerca de 24 milhões de palavras em português brasileiro, criado pelo projeto Processamento Computacional do Português com base nos textos do jornal Folha de São Paulo que fazem parte do *corpus* NILC/São Carlos, compilado pelo Núcleo Interinstitucional de Lingüística Computacional (NILC). Para efeito de teste do sistema, foram utilizados dois extratos deste conjunto (dois conjuntos de quatro classes), visto que o mesmo é demasiado grande, o que demandaria um tempo considerável de processamento.

5.2 O Problema

O problema consiste na implementação de dois modelos computacionais, baseados nos algoritmos bayesiano e de ranqueamento linear propostos por Weiss et al. [63]. Estes modelos são capazes de identificar a classe de textos desconhecidos, a partir do conhecimento obtido de textos previamente classificados.

5.3 Procedimentos para a Solução do Problema

A solução para este problema, naturalmente, envolve a categorização de textos. Cada texto é associado a um vetor de características derivado do próprio texto. As características são palavras que ocorrem no texto, onde cada uma tem um valor numérico baseado na sua freqüência de ocorrências, ou um valor booleano indicando a presença ou ausência da palavra. Um conjunto grande de textos tende a produzir um vocabulário demasiadamente grande, gerando um conjunto de características ainda maior. Todavia, geralmente os textos podem ser classificados a partir apenas de um pequeno conjunto destas características. O vetor correspondente ao texto é submetido a um categorizador. O categorizador por sua vez determina se o texto pertence ou não a determinada categoria que estiver sendo procurada.

Ao final do processo, todos os textos classificados como positivos são coletados e assinalados à sua respectiva classe. Os categorizadores são obtidos usando um algoritmo de aprendizado de máquina que processa um conjunto de vetores de características dos textos para que sejam assinalados à classe correta. Este processo é ilustrado na figura 5.1.

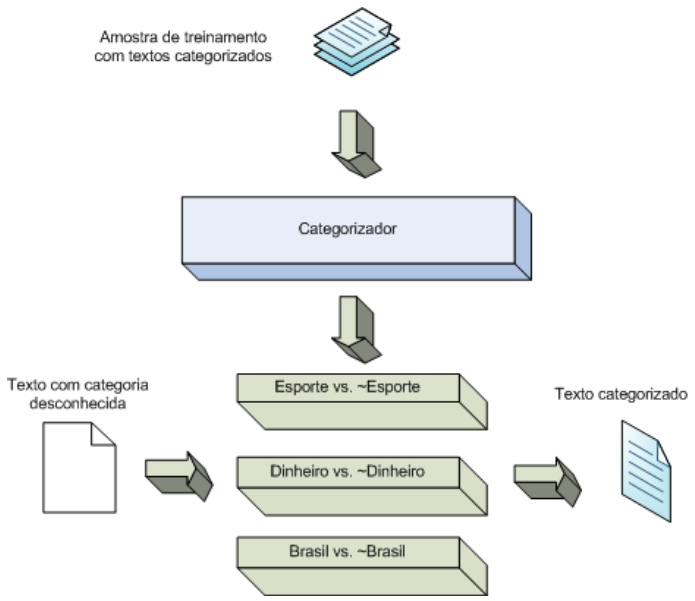


Figura 5.1: Ilustração esquemática do processo de mineração de textos

Pela perspectiva de mineração de textos, os dois aspectos-chave da solução são: (i) geração das características; (ii) uso de classificadores binários.

5.3.1 Teste sem balanceamento

O primeiro teste utiliza um conjunto de textos para treinamento desbalanceados, ou seja, nenhuma relação treinamento:teste foi implementada. A base de treinamento foi obtida de forma aleatória, conforme apresentado na tabela 5.1.

Tabela 5.1: Características do *corpus* "CETENFolha sem balanceamento"

Assunto	Treinamento	Teste
Brasil	2483	100
Dinheiro	2124	100
Esporte	2594	100
Mundo	1565	100

5.3.2 Teste com balanceamento

As aplicações para mineração de textos envolvem algumas questões cruciais que precisam ser analisadas. Questões como qual a quantidade de textos que devem ser usados para o treinamento, quais características são mais apropriadas para os diversos tipos de textos, qual classificador é o mais apropriado e quando os classificadores devem ser retreinados, devem ser respondidas no contexto do problema a ser analisado, não existindo assim, uma regra que defina os melhores parâmetros para a obtenção dos melhores resultados.

Para a realização dos testes, os dados textuais foram divididos em dois conjuntos distintos de treinamento e teste. Foi utilizada uma relação 4:1 entre os textos de treinamento e teste, com características balanceadas, conforme os dados mostrados na tabela 5.2.

Ainda para este teste, sabendo-se que os textos possuem uma ordem cronológica, chegou-se à seguinte questão: Como a performance dos classificadores pode diminuir ao longo do tempo?

Um modelo de classificação treinado com documentos de um período de tempo t_1 a t_2 provavelmente apresentará bons resultados com documentos do período t_3 ($t_3 > t_2$), porém, certamente, apresentará piores resultados com outros documentos do tempo t_n, com $n > 3$, com a performance do sistema decaindo proporcionalmente à medida que n é incrementado. O entendimento desta degradação pode determinar quando deve ser realizado o retreinamento do classificador. Foram realizados alguns testes para avaliar o impacto do tempo para uma amostra específica.

Uma vez que os textos estão divididos por semestre e sabendo-se que cada um possui um identificador seqüencial, foi assumido que este identificador segue a ordem cronológica de publicação dos textos. Por isso os dados foram particionados em seis grupos, com cada grupo contendo textos seguindo a ordem de seus identificadores. O último grupo foi usado como teste, e os modelos de classificadores foram gerados baseados em cada um dos outros cinco grupos, sendo gerado um classificador para cada um deles. Desta maneira, o primeiro conjunto de treinamento contém 1000 textos, sendo 250 para cada classe, o segundo conjunto de treinamento contém 2000 textos, ou seja, 500 para cada classe, e assim por diante até que se atinja 1250 textos por classe, totalizando 5000 textos no último grupo. Cada conjunto possui todos os textos do conjunto anterior de modo a tornar os testes homogêneos.

O objetivo central é realizar o teste do ambiente de *grid* computacional. Um segundo

objetivo é observar como a performance dos classificadores varia com diferentes cenários do passado, para se obter o número ideal de textos para treinamento. Os assuntos disponíveis nestes textos são mostrados na tabela 5.2.

Tabela 5.2: Características do *corpus* CETENFolha com balanceamento

Assunto	Treinamento Máximo	Teste
Brasil	1250	250
Dinheiro	1250	250
Esporte	1250	250
Mundo	1250	250

5.3.3 Padronizações do Textos para Uso no Ambiente

Os textos CETENFolha são fornecidos como um único conjunto em formato *XML*. Com o objetivo de tornar este conjunto manipulável pelo portal Aîuri, foi necessária a conversão dos textos para arquivos individuais com formato ASCII.

A leitura automática de textos no formato XML não foi implementada porque, para isso, é necessário que o usuário tenha conhecimento dos metadados associados, o que nem sempre é possível.

Cada texto deste arquivo possui uma quantidade considerável de metadados associados, como identificador do texto, título, o próprio texto, semestre, caderno no jornal etc. Para a realização dos testes foi dado foco no metadado "cad" que, de fato, especifica a qual categoria pertence o texto. Foi desenvolvida uma aplicação para ler o arquivo *XML* e realizar a tarefa de retirar os metadados. Isto demandou um trabalho significativo a nível de programação, uma vez que a estrutura do arquivo original, por vezes, não seguia o padrão hierárquico esperado. Para ambos os conjuntos, os arquivos devem seguir o padrão, conforme a tabela 5.3.

Tabela 5.3: Padrão de nomenclatura dos arquivos

Treino	Teste
Classe-nomedoarquivo.txt	[Classe-]nomedoarquivo.txt

As composições Classe/nomedoarquivo não podem conter acentos. O item "Classe" é obrigatório para os arquivos que compõem o conjunto de treinamento e opcional para o conjunto de teste. A primeira linha de ambos os tipos de arquivos deve conter o título dos

textos. Na ausência do título deve ser colocado qualquer caracter do alfabeto. Para estes casos o padrão adotado foi a *string* "xxx". Na tabela 5.4 são ilustrados os dois formatos possíveis de arquivos.

Tabela 5.4: Exemplos de arquivos manipuláveis pelo portal

Arquivo	Texto
Esporte-15.txt	A batalha da Fifa SÍLVIO LANCELLOTTI Começou bem antes do que se previa a batalha pela futura sucessão na Fifa apenas seis meses depois do super-acordo que, nas vésperas da Copa, reconduziu o brasileiro João Havelange ao sexto mandato consecutivo . Pelo acordo, os três continentes mais obstinados em cortar o reinado de Havelange, a África, a Ásia e a Europa, aceitaram cancelar os seus movimentos de oposição em troca, basicamente, de dois compromissos . Havelange aceitaria engolir o italiano Antonio Matarrese como o seu vice-executivo e, além disso, esqueceria os seus modos autoritários, coordenando a entidade de maneira colegiada .
teste.txt	xxx A pesquisa mais recente do Datafolha coloca, por ora, Fernando Henrique Cardoso (PSDB) como o segundo colocado . Mas a convenção dos «tucanos», que seria mera formalidade, agora promete uma disputa, já que o ex-governador da Bahia e atual deputado federal, Waldir Pires, anuncia a sua inscrição para enfrentar FHC . Suas chances parecem mais do que remotas, mas de qualquer forma cria um constrangimento .

Desta maneira, ambos os conjuntos estão prontos para serem convertidos para o formato *XML* e serem utilizados pelo sistema Aîuri para a geração dos modelos.

5.3.4 Avaliação dos Categorizadores

A seguir são avaliados os resultados gerados pelos categorizadores implementados na ferramenta proposta. Dois tipos de testes foram realizados com os mesmos dados em dois ambientes disponíveis.

O primeiro deles foi na máquina local, ou seja, a execução foi realizada na própria máquina do portal, com todos os dados necessários para processamento já disponíveis na mesma.

O segundo foi a submissão dos dados no *grid* NACAD, onde de fato, estão disponíveis como serviço em cada nó do *grid*, métodos que chamam os algoritmos categorizadores. Os dados necessários para o processamento, ou seja, os arquivos de treinamento e teste em formato *XML*, o arquivo de *stoplist* e os *stems* devem ser carregados para o nó de processamento

no momento da chamada do *serviço web*. Também, neste caso, os resultados gerados, ou seja, o arquivo em formato *XML* com os textos classificados positivamente, assim como os classificados de forma negativa e o arquivo com as métricas obtidas ao final do processamento são copiados para a máquina do portal para o diretório do usuário que fez a chamada. Todas estas atividades envolvem transferência de dados via rede, o que, dependendo do tamanho dos arquivos envolvidos, e da capacidade da rede, obviamente gera um retardo esperado.

O portal Aîuri disponibiliza diferentes ambientes de execução que podem ser avaliados por seus usuários, permitindo a este a escolha do melhor ambiente possível, conforme a complexidade de seu experimento.

Também, uma vez que os algoritmos submetidos nos dois ambientes são iguais, os resultados produzidos também são idênticos, variando somente os tempos de processamento, conforme os testes que foram realizados.

A medida utilizada para avaliar o desempenho dos classificadores foi a *f-measure*, que é obtida a partir dos textos do conjunto de teste, já discutida no capítulo 2. Esta medida combina os valores das medidas de *precisão* e *abrangência*, de maneira a se obter o desempenho geral do sistema, permitindo um balanceamento entre os dois valores. A *f-measure* mede a capacidade de generalização do modelo gerado, permitindo verificar se durante o treinamento este assimilou características significativas do conjunto de treinamento que permitem um bom desempenho em outros conjuntos de dados ou se ocorreu *overfitting*, que é um fenômeno no qual o modelo especializa-se nos dados de treinamento. O valor da *f-measure* é maximizado quando a *precisão* ou *abrangência* são iguais ou muito próximas.

Deve-se lembrar, que em geral, a eficiência da extração das entidades de um texto se comporta de forma exponencial à complexidade algoritmica. É possível, com heurísticas simples, conforme foi realizado nos testes que seguem, atingir facilmente um nível de acerto de 80%. A dificuldade consiste em se obter valores acima de 90%.

5.3.4.1 Categorizador *Bayesiano*

Os classificadores bayesianos são classificadores estatísticos, baseados no teorema de *Bayes*, capazes de estimar a qual classe determinado registro de uma amostra pertence. No presente teste, no contexto de mineração de textos, foram utilizados os conceitos do classificador *bayesiano* ingênuo, no qual as variáveis são consideradas independentes entre si. Esta

suposição simplifica a abordagem do problema de classificação sem comprometer significativamente a precisão do resultado.

Nos testes foram utilizadas quatro abordagens. A primeira delas engloba a geração do classificador sem a utilização de *stopwords* e *stems*, variando somente o número de termos do dicionário. Na segunda abordagem foi incluído o uso de *stopwords*. Na terceira abordagem foram utilizados *stopwords* e *stems*. O uso somente de *stems* foi aplicado na quarta abordagem.

Observou-se que para a maioria das classes quando houve a utilização de *stopwords* e *stems*, ocorreram melhorias nos resultados obtidos, porém, para o caso dos *stems*, houve um grande aumento no tempo de processamento, conforme as medidas obtidas nos experimentos.

Os parâmetros *probability-threshold* e *reject-threshold* foram alterados, com diversas combinações, sendo observadas algumas variações nos resultados. Quando foi atribuído o valor 1 para o parâmetro *reject-threshold*, verificou-se que o classificador obteve *f-measure* nulo, ou seja, todos os textos foram rejeitados, o que também já era esperado.

Uma questão importante foi a escolha do tamanho dos dicionários. A mais importante característica de um dicionário é o seu tamanho. Fatores como os métodos de predição utilizados e a capacidade computacional, são primordiais para a definição do tamanho do dicionário. Foram realizados testes com dicionários de até 5000 termos.

Sabe-se que um dicionário é construído baseado na coleção de termos existentes no conjunto de treinamento. Para esta tarefa, pode-se usar *stopwords*, que consistem em um arquivo que contém uma série de palavras que não possuem nenhum significado semântico, como artigos, preposições etc. Uma outra técnica que pode ser utilizada é a implementação de *stems*, que consiste em um arquivo contendo palavras do conjunto de treinamento reduzidos ao seu radical. A utilização de *stopwords* e *stems* tende a reduzir significativamente o tamanho do dicionário gerado, porém algumas considerações devem ser feitas em relação ao uso de *stems*. Apesar de contribuir de forma relevante para a redução do tamanho do dicionário, seu uso deve ser criteriosamente avaliado, uma vez que acarreta um alto custo computacional. Portanto, o uso de *stems* é indicado somente se os benefícios obtidos forem realmente relevantes. Em todos os testes realizados seu uso gerou um aumento considerável no tempo de processamento, até para pequenos conjuntos de treinamento, sem produzir uma melhora relevante nos modelos gerados.

Outro cuidado que deve ser observado é que dicionários muito grandes tendem a gerar *overfitting*, apresentando baixo desempenho para classificar textos desconhecidos. Dicionários menores, além de evitarem o *overfitting*, reduzem o tempo de processamento dos algoritmos, fato que pode ser observado nos experimentos realizados.

Alguns ajustes na aplicação foram necessários para fins de otimização de processamento, uma vez que os objetos que representam as várias entidades do sistema eram criados e não estavam sendo retirados da memória pela *Garbage Collector* da *Java Virtual Machine* (*JVM*) quando não eram mais necessários. Isto ocorreu pelo fato de que a *Garbage Collector*, que é o componente responsável pela retirada da memória de objetos que não estão sendo mais referenciados, retirando do programador a responsabilidade por destruí-los, ser uma *thread* de baixa prioridade e somente é executada quando a *CPU* está livre. Foi observado que durante o processamento dos testes, a *CPU* teve 100% de utilização, não permitindo assim que os objetos sem uso fossem retirados, não liberando, desta maneira, memória para processamento. A solução para o problema foi fazer chamadas explícitas à *Garbage Collector* em locais específicos da aplicação. Feito isto, foi observado um incremento relevante na capacidade de processamento do ambiente, mantendo um uso praticamente constante de memória durante todo o tempo de execução.

Nos gráficos da figura 5.2 são ilustrados os resultados obtidos nos testes realizados com desbalanceamento.

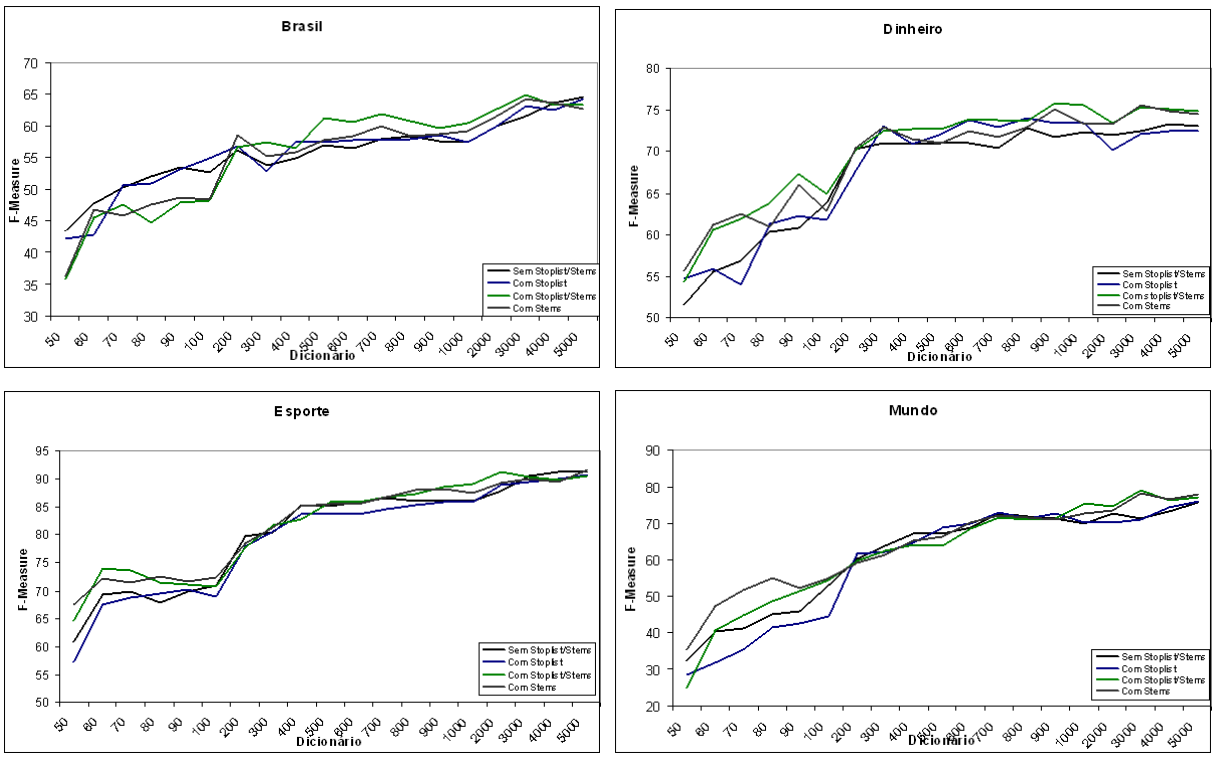


Figura 5.2: Resultados do Categorizador Bayesiano para os testes com desbalanceamento

Pela observação dos resultados obtidos para a classe “Brasil”, foi verificada a dificuldade do classificador para encontrar termos relevantes para esta classe, apresentando uma média de 55,57% para a *f-measure*. A melhor *f-measure* obtida foi de 64,94% com a utilização de *stoplist* e *stems*, com 3000 termos no dicionário e um tempo de processamento de 21 seg. na máquina local e 23 seg. no *grid*. Porém, com a utilização somente de *stoplist* e 5000 termos no dicionário, foi obtida uma *f-measure* de 64,24% em 4 seg. na máquina do local e 4 seg. no *grid*.

Os resultados obtidos para a classe “Dinheiro” foram superiores aos classe “Brasil”, apresentando uma média de 68,71% para a *f-measure*. A melhor *f-measure* obtida foi de 75,57% com a utilização de *stems*, 3000 termos no dicionário e um tempo de processamento de 22 seg. na máquina local e 22 seg. no *grid*. Porém, com 800 termos no dicionário e com a utilização de *stoplist*, foi obtida a *f-measure* de 74,04%, em apenas 3 seg. na máquina local e 3 seg. no *grid*.

Para a classe “Esporte”, foi observado que o classificador produziu os melhores resultados obtidos. O bom nível de qualidade do modelo para esta classe também pode ser justificado por ser esta a que possui o maior conjunto de textos para treinamento e o seu

vocabulário ser fortemente diferenciado das demais classes, fazendo com que o classificador tenha mais facilidade de encontrar termos relevantes. Foi obtida a *f-measure* de 91,54%, em 21 seg. na máquina local e 21 seg. no *grid*, com a utilização de *stems* e 5000 termos no dicionário. O tempo de 4 seg. na máquina local e 4 seg. no *grid* foi verificado, para a *f-measure* de 90,62%, com o uso de *stoplist* e, também, 5000 termos no dicionário.

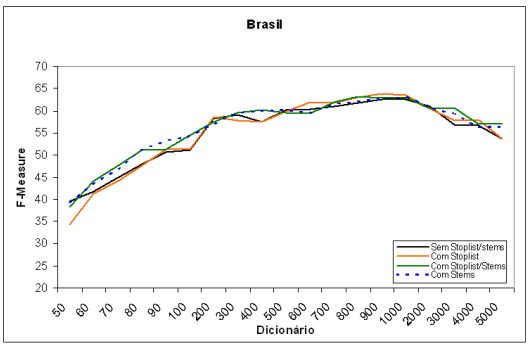
Do mesmo modo que nas classes anteriores, com a classe "Mundo" foi obtidos resultados melhores quando foram utilizados *stems*. A melhor *f-measure* obtida foi de 79,1%, com 3000 termos no dicionário, com o uso de *stoplist* e *stems*, no tempo de 21 seg. na máquina local e 21 seg. no *grid*. Porém, com 5000 termos no dicionário e com a utilização de *stoplist*, foi obtida a *f-measure* de 75,9%, em apenas 4 seg. na máquina local e 4 seg. no *grid*.

A seguir são apresentados os resultados obtidos nos testes com balanceamento. Nos gráficos da figura 5.3 são ilustrados os resultados para a classe "Brasil".

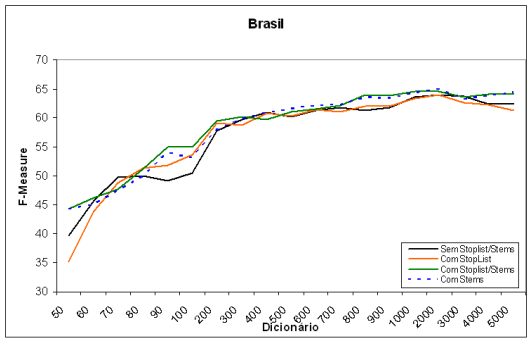
Para esta classe, observa-se que este classificador apresentou padrões de resultados semelhantes, com um conjunto de treinamento com 500 termos ou mais. Observa-se também que a presença ou ausência de *stopwords* e *stems* pouco influenciou na qualidade do modelo gerado. Verifica-se a dificuldade deste classificador para encontrar termos relevantes para esta classe, apresentando a *f-measure* em torno de 60%. Observa-se que, neste caso, o conjunto de treinamento com 750 textos e 4000 termos no dicionário apresenta resultados satisfatórios em relação aos demais, obtendo a *f-measure* de 67,12%, usando *stoplist* e *stems*, em 7 seg. na máquina local e 8 seg. no *grid*. Também foi observado que a diferença entre esta medida e a melhor avaliação registrada, com 1250 textos para treinamento e dicionário com 3000 termos usando *stems*, com a a *f-measure* de 68,57%, foi de 5,89%. Os tempos obtidos para esta medida foram de 10 seg. na máquina local e 10 seg. no grid.

Nos gráficos da figura 5.4 são ilustrados os resultados para a classe "Dinheiro".

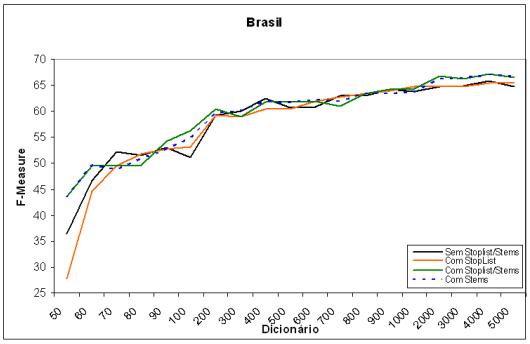
Este classificador, para a classe "Dinheiro", apresentou resultados semelhantes para todos os conjuntos de treinamento com 500 textos ou mais. Observa-se que o conjunto de treinamento com 1000 textos e um dicionário com 5000 termos usando *stopwords*, gerou a *f-measure* de 78,71% em 2 seg. na máquina local e 2 seg. no *grid*. Assim como na classe anterior, este classificador apresenta dificuldade em estabelecer um vocabulário que defina esta classe. O melhor modelo obtido foi com 1250 textos no conjunto de treinamento, usando 5000 termos no dicionário, com *stems*, que apresentou a *f-measure* de 79,53%. O tempo obtido foi de 10 seg. na máquina local e 9 seg. no *grid*.



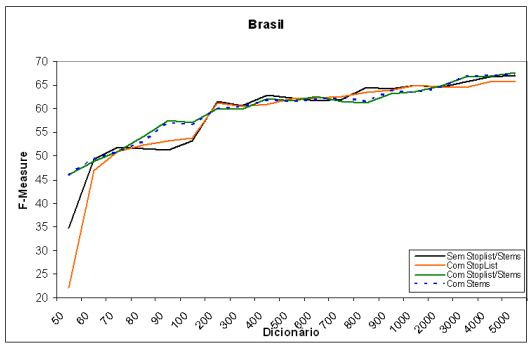
(a) Treinamento com 250 textos



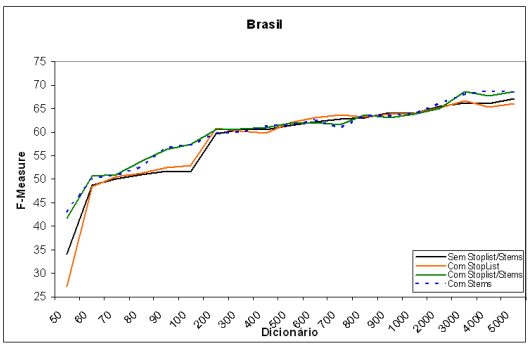
(b) Treinamento com 500 textos



(c) Treinamento com 750 textos



(d) Treinamento com 1000 textos



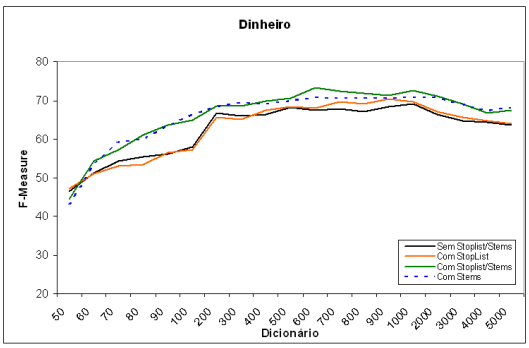
(e) Treinamento com 1250 textos

Figura 5.3: Resultados do Categorizador Bayesiano para a classe “Brasil” com balanceamento

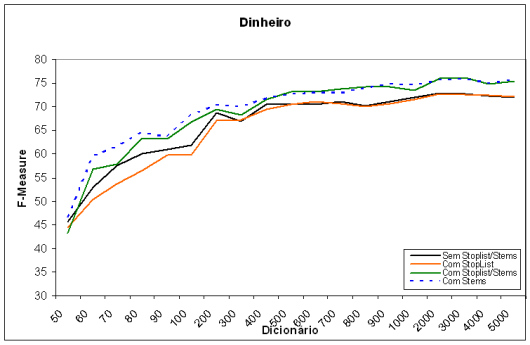
Nos gráficos da figura 5.5 são apresentados os resultados para a classe “Esporte”.

Observa-se que o melhor modelo obtido para a classe “Esporte” foi com a utilização de 1250 no conjunto de treinamento, 3000 termos no dicionário, usando *stoplist* e *stems*. O tempo obtido foi de 9 seg. na máquina local e 9 seg. no *grid*. Com o uso somente de *stoplist*, a *f-measure* foi 91,18% em 2 seg. na máquina local e 3 seg. no *grid*.

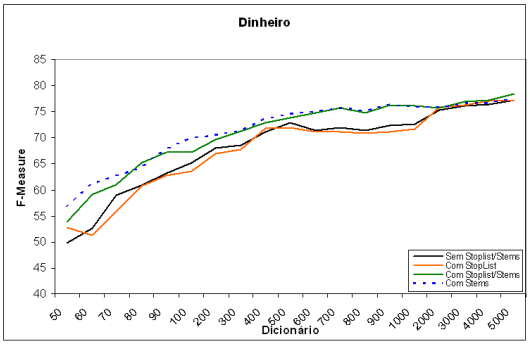
Pode-se explicar a evidente superioridade desta classe em relação às demais, pelos termos altamente diferenciados que compõem os textos da classe “Esporte”, em muito se diferenciando em relação às outras classes, que por sua vez possuem termos que tendem a gerar



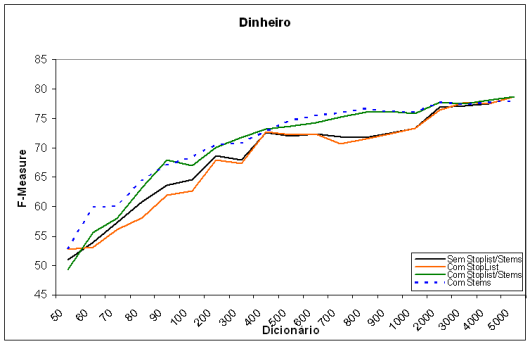
(a) Treinamento com 250 textos



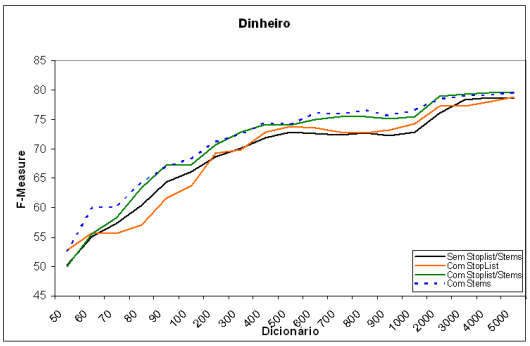
(b) Treinamento com 500 textos



(c) Treinamento com 750 textos



(d) Treinamento com 1000 textos

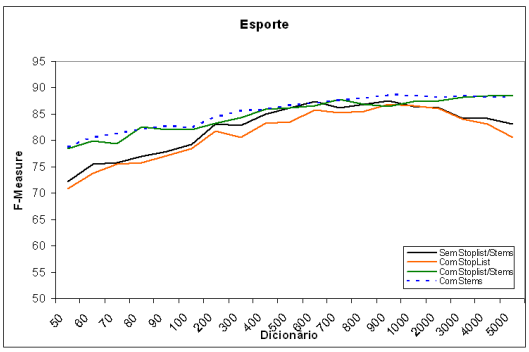


(e) Treinamento com 1250 textos

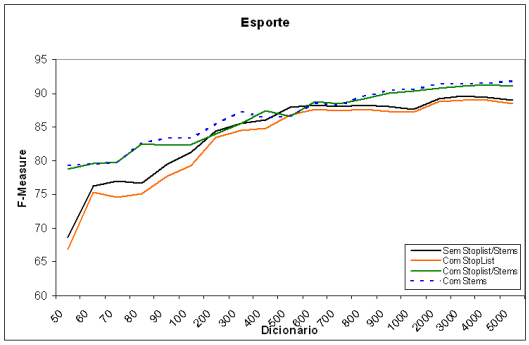
Figura 5.4: Resultados do Categorizador Bayesiano para a classe “Dinheiro” com balanceamento

modelos com menor desempenho. Isto ocorre pelo fato de que as demais classes possuem muitos termos em comum em seus textos, gerando dificuldades para o classificador determinar seus termos relevantes.

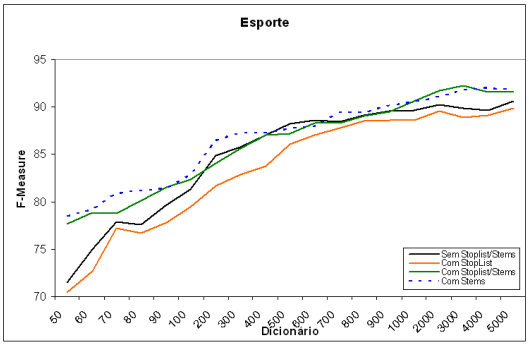
Nos gráficos da figura 5.6 são ilustrados os resultados para a classe “Mundo”. A melhor classificação obtida para a classe “Mundo” foi utilizando 1000 textos no conjunto de treinamento, 4000 termos no dicionário, com *stoplist* e *stems*, apresentando a *f-measure* de 81,78%, em 7 seg. na máquina local e 8 seg. no *grid*. Somente com a utilização de *stoplist* o tempo de processamento foi de 2 seg. na máquina local e 2 seg. no *grid*.



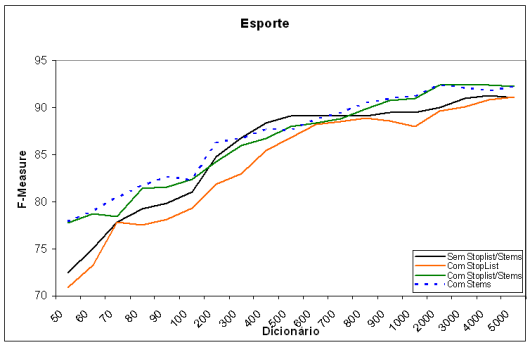
(a) Treinamento com 250 textos



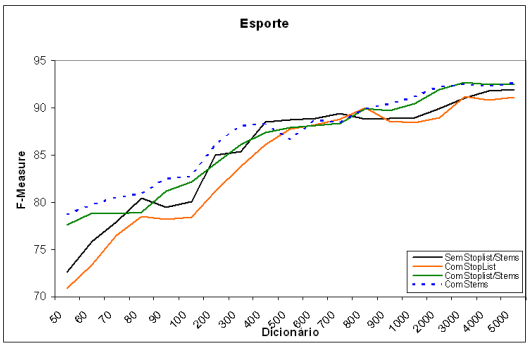
(b) Treinamento com 500 textos



(c) Treinamento com 750 textos



(d) Treinamento com 1000 textos



(e) Treinamento com 1250 textos

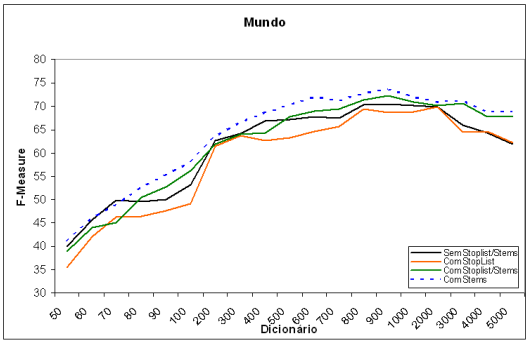
Figura 5.5: Resultados do Categorizador Bayesiano para a classe Esporte com balanceamento

5.3.4.2 Categorizador Linear

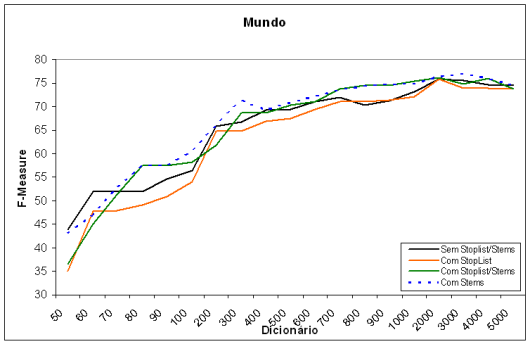
Os categorizadores baseados em métodos lineares são uma abordagem clássica para a solução de problemas de predição.

O método *bayesiano*, utilizado anteriormente, pode ser interpretado como um caso especial do método linear, porém tem graves problemas com atributos redundantes, pois funciona melhor quando lida com um pequeno número de atributos, fato que não ocorre quando gera-se dicionários com milhares de palavras.

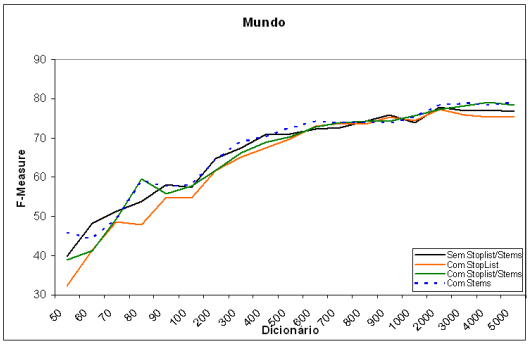
Uma vantagem da abordagem linear é que a geração do modelo é muito simples, desde



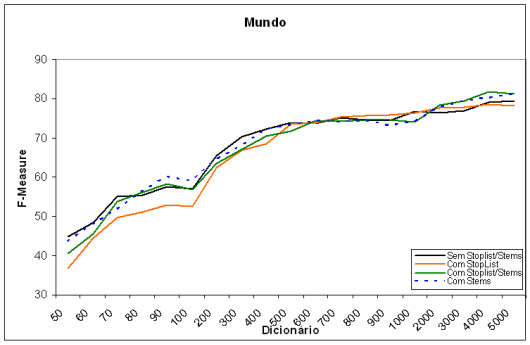
(a) Treinamento com 250 textos



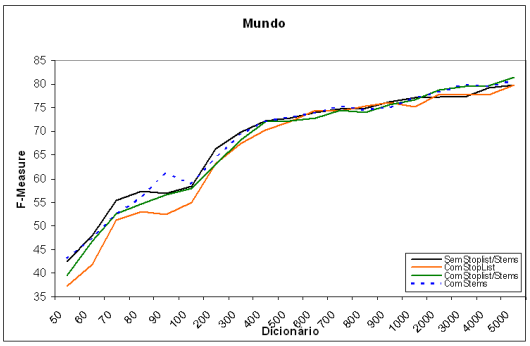
(b) Treinamento com 500 textos



(c) Treinamento com 750 textos



(d) Treinamento com 1000 textos



(e) Treinamento com 1250 textos

Figura 5.6: Resultados do Categorizador Bayesiano para a classe Mundo com balanceamento

que seja feita uma escolha de expressões úteis e que o algoritmo de aprendizado seja capaz de determinar os pesos para cada expressão criada.

A abordagem utilizada é idêntica à da seção 5.3.4.1, com o mesmo conjunto de treinamento e teste.

Alguns testes foram realizados com o parâmetro *decision-threshold*, que de acordo com os seus valores prioriza a precisão (*precision*) ou a recordação (*recall*).

A priorização da precisão é adequada quando se sabe exatamente qual a classe procurada, de maneira que os resultados retornados são mais precisos. Já a priorização da recordação é mais adequada quando as classes não são conhecidas, de modo que o pesquisador ainda ne-

cessita fazer um reconhecimento do domínio dos dados, obtendo como retorno uma elevada quantidade de textos, ainda que fora de sua classificação correta.

Nos testes realizados, foi observado que com o reforço da precisão foram retornados poucos documentos, porém com um maior nível de acertos. O reforço da recordação gerou um retorno muito maior de documentos, porém com um maior número de classificações erradas.

Na prática existe um *trade off* entre precisão e recordação. Ao maximizar a precisão, reduz-se a recordação. Ao maximizar a recordação, reduz-se a precisão. No entanto, o principal objetivo é manter os valores das duas medidas altas, reforçando uma ou outra dependendo da funcionalidade da aplicação. Se são retornados muitos documentos, certamente a recordação está maximizada em detrimento da precisão. Provavelmente existem quantidades significativas de textos classificados incorretamente. Porém, se é retornado somente um documento, e classificado corretamente, a precisão está maximizada em detrimento da recordação. Logo, se o objetivo for uma investigação sobre os textos, deve-se reforçar a recordação, contudo, se o objetivo já estiver definido pode-se reforçar a precisão.

Nos gráficos da figura 5.7 são apresentados os resultados obtidos nos testes realizados com desbalanceamento.

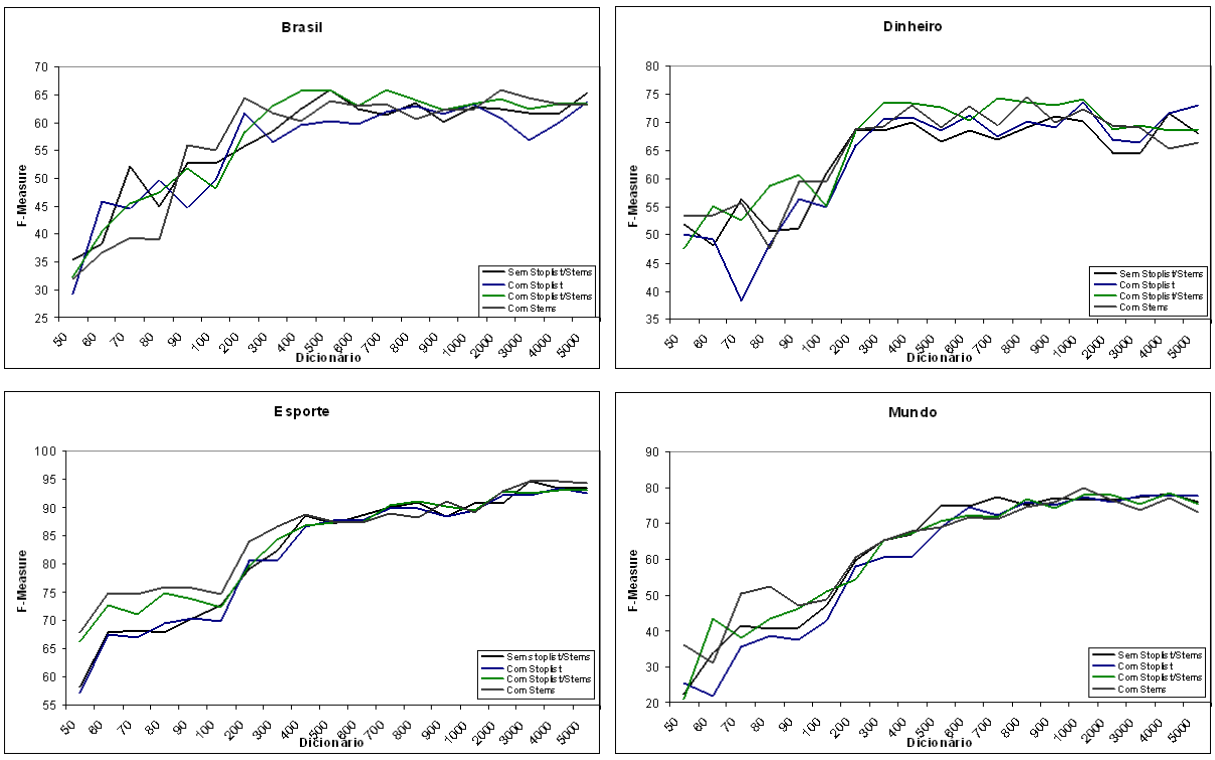


Figura 5.7: Resultados do Categorizador Linear para os testes com desbalanceamento

Observa-se que para a classe “Brasil”, o classificador obteve a *f-measure* média de 56,59%. O melhor modelo encontrado para esta classe foi com 2000 termos no dicionário, usando *stems*, atingindo a *f-measure* de 65,99% em 21 seg. na máquina local e 23 seg. no *grid*. O resultado mais próximo sem a utilização de *stems* foi verificado com o uso de *stoplist*, 5000 termos no dicionário e a *f-measure* obtida é de 63,78%. Este resultado foi gerado em 4 seg. na máquina local e 5 seg. no *grid*.

Para a classe “Dinheiro”, os resultados foram ligeiramente superiores. Com o dicionário com 800 termos, usando *stems*, foi obtida a *f-measure* de 74,63% em 21 seg. na máquina local e 22 seg. no *grid*. A melhor medida gerada sem a utilização de *stems*, foi com 1000 termos no dicionário, usando *stoplist*, obtendo *f-measure* de 73,57%. Esta medida foi obtida em 3 seg. na máquina local e 4 seg. no *grid*.

A classe “Esporte”, da mesma maneira que no classificador bayesiano, atingiu os melhores resultados em relação às demais classes. Foi obtida a *f-measure* de 94,79% em 21 seg. na máquina local e 23 seg. no *grid*, com o uso de *stems* e o dicionário com 3000 termos. A *f-measure* de 93,4%, com 4000 termos no dicionário usando somente *stoplist*, foi obtida em 4 seg. na máquina local e 5 seg. no *grid*.

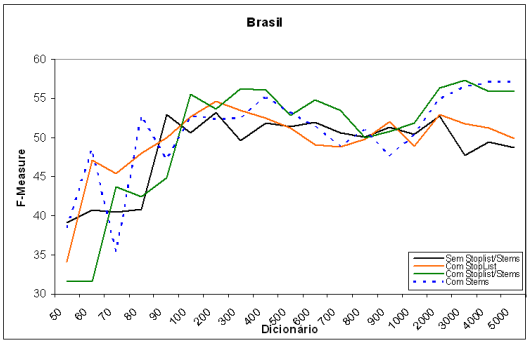
Da mesma maneira que nas classes anteriores, com a classe “Mundo” foram obtidos resultados superiores quando houve a utilização de *stems*. A melhor medida obtida foi de 80%, com 1000 termos no dicionário, usando somente *stems*. Este resultado foi gerado em 21 seg. na máquina local e 22 seg. no *grid*. A medida mais próxima sem o uso de stems foi de 77,71%, com o dicionário com 3000 termos, somente utilizando *stoplist*. Este resultado foi obtido em 4 seg. na máquina local e 4 seg. no *grid*.

Nos gráficos da figura 5.8 são ilustrados os resultados para a classe “Brasil”.

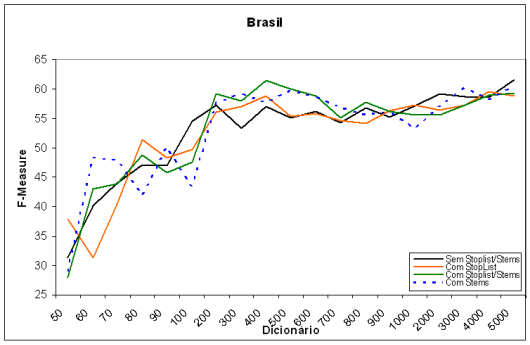
A melhor classificação obtida para a classe “Brasil” foi com 1250 textos no conjunto de treinamento, 3000 termos no dicionário, usando *stoplist* e *stems*. Para esta configuração a *f-measure* foi de 64,39% em 10 seg. na máquina local e 10 seg. no *grid*. Usando somente *stoplist* a *f-measure* foi 62,03% em 3 seg. na máquina local e 3 seg. no *grid*.

Nos gráficos da figura 5.9 são ilustrados os resultados para a classe “Dinheiro”.

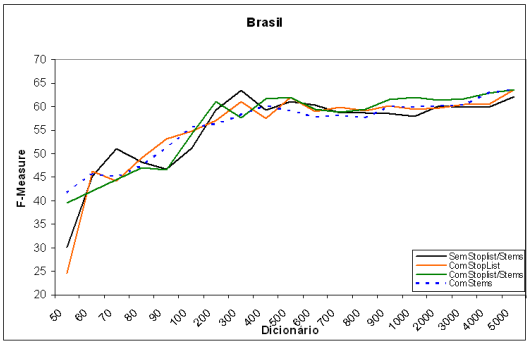
O comportamento deste classificador para a classe “Dinheiro” foi bem heterogêneo conforme as diferentes configurações. A melhor classificação foi obtida com 1250 textos no conjunto de treinamento e 500 termos no dicionário, com *stems*, onde chegou-se a 74,13% na *f-measure* em 9 seg. na máquina local e 10 seg. no *grid*. Usando somente *stoplist* a



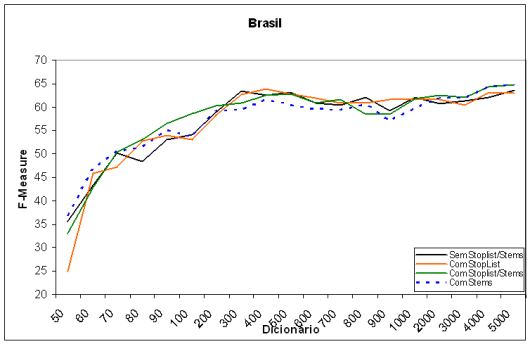
(a) Treinamento com 250 textos



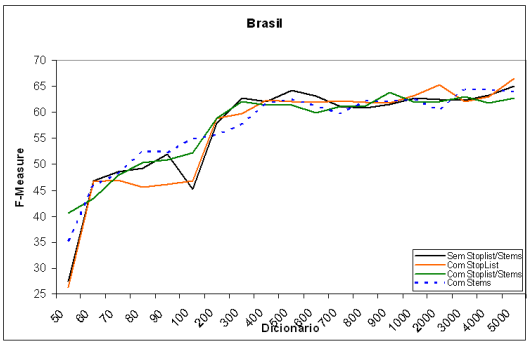
(b) Treinamento com 500 textos



(c) Treinamento com 750 textos



(d) Treinamento com 1000 textos



(e) Treinamento com 1250 textos

Figura 5.8: Resultados do Categorizador Linear para a classe Brasil com balanceamento

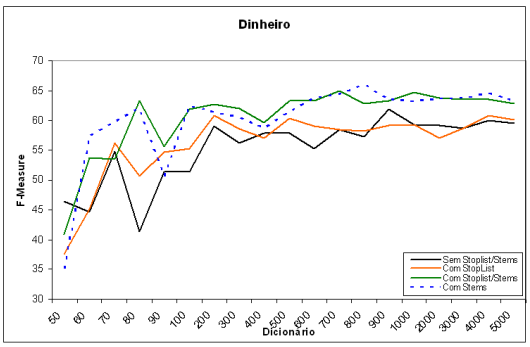
medida foi 71,18% em 2 seg. na máquina local e 2 seg. no *grid*.

Nos gráficos da figura 5.10 são apresentados os resultados para a classe “Esporte”.

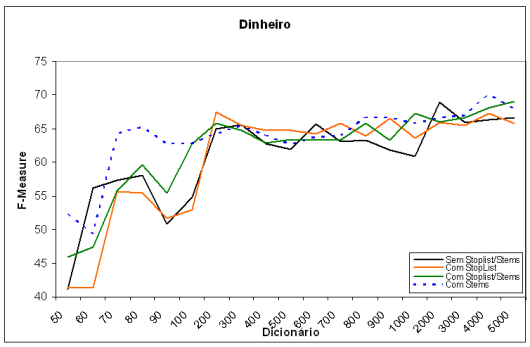
Da mesma forma que no classificador *bayesiano*, com a classe “Esporte” foram obtidos excelentes resultados para todos os experimentos. A utilização de 1250 textos no conjunto de treinamento gerou bons resultados com 5000 termos no dicionário, usando *stems*, atingindo a *f-measure* de 87,83% em 9 seg. na máquina local e 10 seg. no *grid*. Com o uso de *stoplist* a medida foi de 86,75% em 3 seg. na máquina local e 3 seg. no grid.

Nos gráficos da figura 5.11 são ilustrados os resultados para a classe “Mundo”.

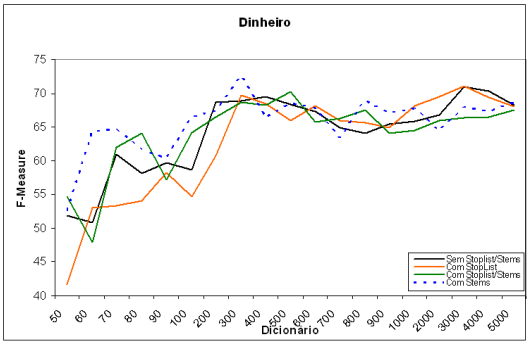
Claramente os melhores resultados para a classe “Mundo” foram obtidos com o conjunto



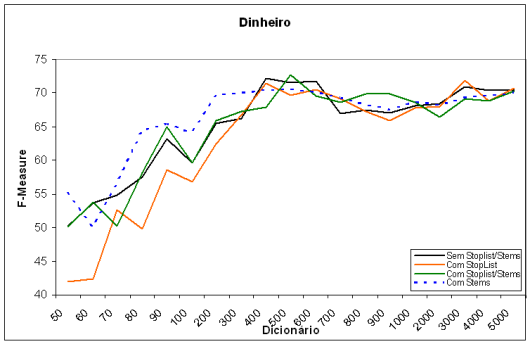
(a) Treinamento com 250 textos



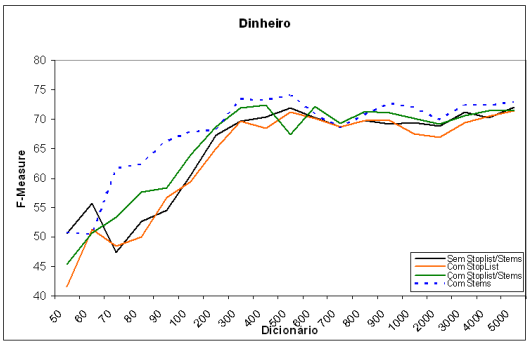
(b) Treinamento com 500 textos



(c) Treinamento com 750 textos



(d) Treinamento com 1000 textos



(e) Treinamento com 1250 textos

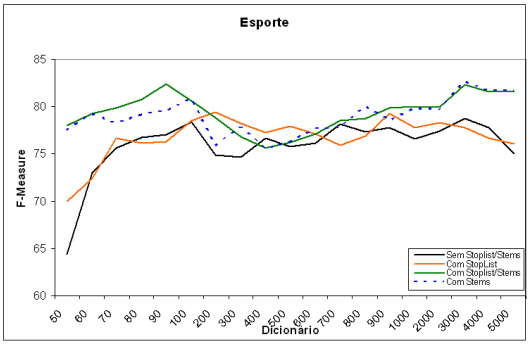
Figura 5.9: Resultados do Categorizador Linear para a classe Dinheiro com balanceamento

de treinamento com 1250 textos, atingindo a *f-measure* de 77,47% com um dicionário de 700 termos usando *stems*, no tempo de 8 seg. na máquina local e 9 seg. no grid.

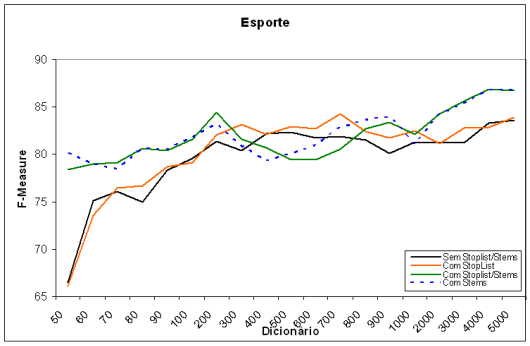
Diante de todos os resultados obtidos, observa-se que o uso de *stems* acarretou um altíssimo custo computacional, não justificando, de uma maneira geral, seu uso, em relação aos benefícios obtidos.

O uso de *stopwords* foi uma boa opção, uma vez que diminuiu o tamanho dos dicionários e apresentou baixo custo computacional.

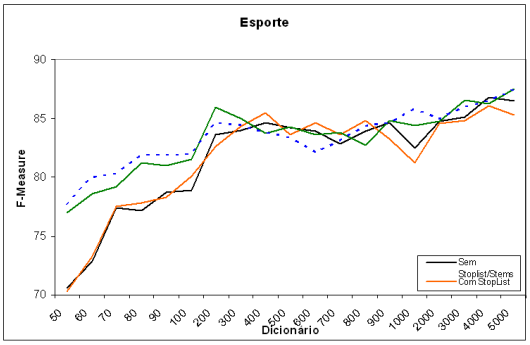
Na figura 5.12 são ilustradas as médias das *f-measures* obtidas para os testes realizados. Os resultados dos testes realizados com os conjuntos de treinamento desbalanceados foram



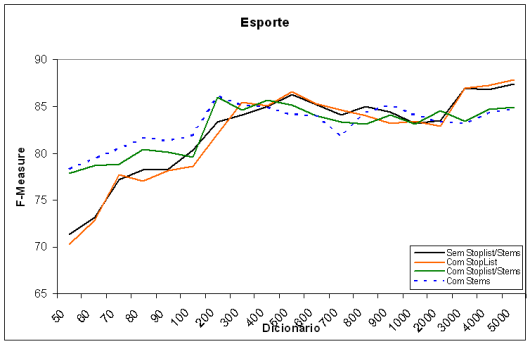
(a) Treinamento com 250 textos



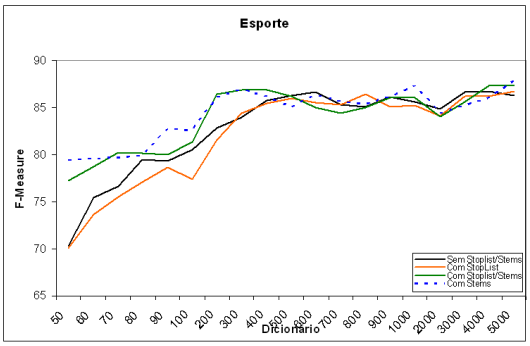
(b) Treinamento com 500 textos



(c) Treinamento com 750 textos



(d) Treinamento com 1000 textos



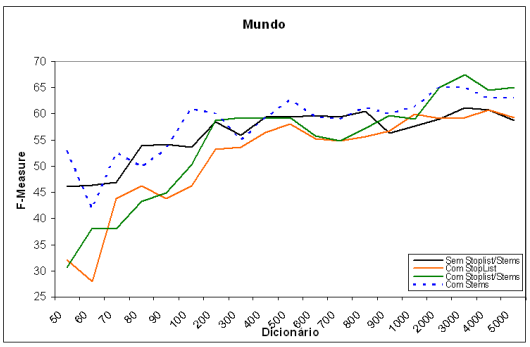
(e) Treinamento com 1250 textos

Figura 5.10: Resultados do Categorizador Linear para a classe Esporte com balanceamento

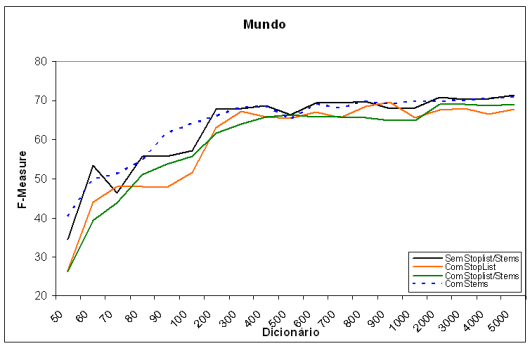
superiores aos realizados utilizando balanceamento, o que pode ser observado comparando-se os resultados ilustrados nos gráficos da figura 5.12.

Com a classe “Esporte”, em todos os casos, foram obtidos melhores modelos, certamente por esta classe possuir um vocabulário que se diferencia do vocabulário das demais classes.

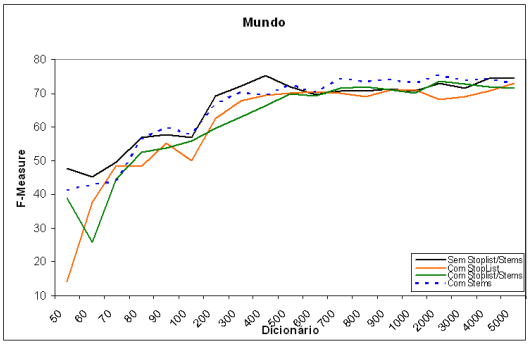
Observa-se, também, que no teste com balanceamento, torna-se difícil identificar quando deve ser realizado o retreinamento do modelo, uma vez que para todas as classes, o classificador manteve o mesmo padrão de comportamento para todas as quantidades de textos no conjunto de treinamento. Daí, pode-se deduzir que o número de textos não foi suficiente para concluir quando é necessário um novo treinamento do modelo, devendo ser implementados



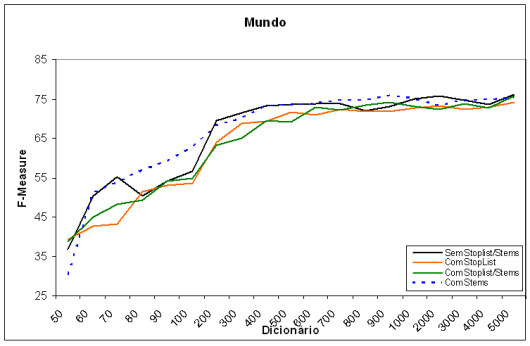
(a) Treinamento com 250 textos



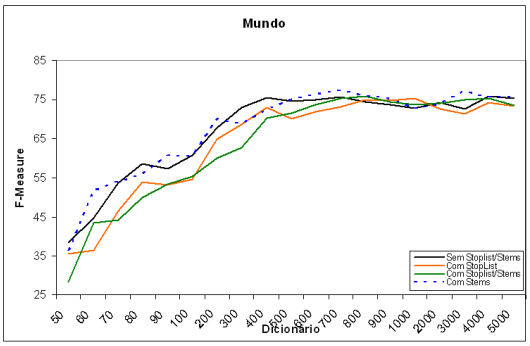
(b) Treinamento com 500 textos



(c) Treinamento com 750 textos



(d) Treinamento com 1000 textos



(e) Treinamento com 1250 textos

Figura 5.11: Resultados do Categorizador Linear para a classe Mundo com balanceamento

novos testes com conjuntos de treinamento maiores.

Deve ser ressaltado que o uso de *stems*, na maioria dos modelos gerados, resultou em melhores medidas. Porém, como pôde ser observado em todos os testes, sua utilização gerou somente pequenas melhoras, o que confirma a afirmação de Weiss et al. [63], que dizem que, no contexto da classificação de textos, o uso de *stems* pode prover, em alguns casos, um pequeno benefício aos modelos gerados. Nos testes realizados, ficou evidente que, a utilização de *stems* não gerou nenhuma melhora quantitativa de acertos que justificasse seu uso. Pode-se observar que em média, sua utilização gerou uma melhora nos modelos em torno de 2%, porém com o aumento do custo computacional em torno de 700%.

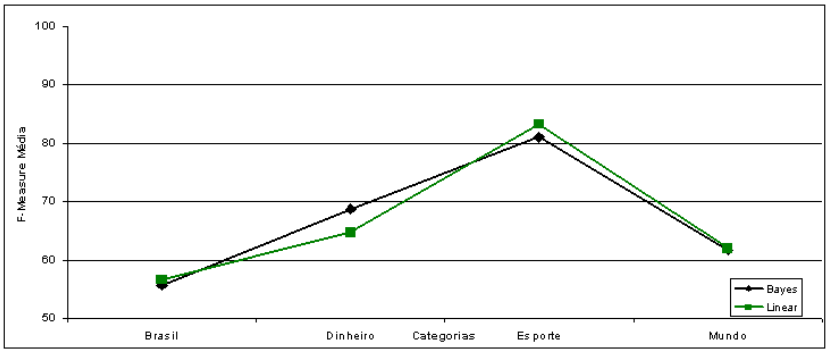
Também, foi observado que para a obtenção das melhores medidas para os modelos, os dicionários ficaram na faixa de 3000 a 5000 termos, que é um número elevado, o que pode gerar *overfitting*.

O comportamento dos algoritmos na máquina local e no *grid* foram idênticos, o que já era esperado, porém no *grid* os tempos de processamento foram um pouco maiores, pois a máquina na qual foram submetidos possui um poder de processamento menor do que a máquina local. As médias dos tempos obtidos nos testes podem ser observadas na tabela 5.5.

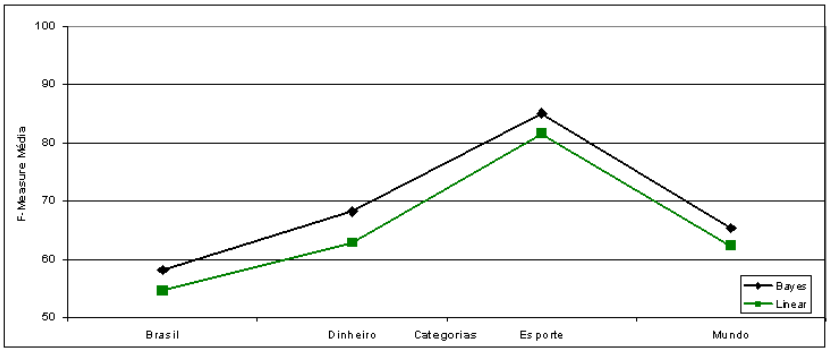
Tabela 5.5: Avaliação geral dos categorizadores

	Máquina do portal	Grid
Sem Stoplist/Stems	3s	4s
Com Stoplist	3s	4s
Com Stoplist/Stems	21s	22s
Com Stems	21s	22s

Logo, da observação de todos os testes realizados, pode-se inferir que ambos os classificadores obtiveram um mesmo padrão de comportamento, com uma performance muito similar. O classificador bayesiano, em média, foi mais rápido, e obteve modelos de classificação superiores aos obtidos pelo classificador linear, sendo, desta maneira, uma boa opção de classificador para os textos em questão, com bom desempenho na máquina local e no *grid*.



(a) *F-Measure* média entre dos categorizadores bayesiano e linear para o teste sem balanceamento



(b) *F-Measure* média entre dos categorizadores bayesiano e linear para o teste com balanceamento

Figura 5.12: Comparação de resultados entre os classificadores

Capítulo 6

Conclusões e Trabalhos Futuros

O objetivo do Projeto Aîuri é a criação de um ambiente cooperativo acadêmico de alto desempenho que tem utilidade no ensino e pesquisa nas áreas de inteligência computacional, análise, avaliação e visualização de dados não-estruturados, com serviços de *grid* para mineração de textos, com a implementação de um portal *web* que receba requisições para serem executadas neste tipo de ambiente.

O sistema Aîuri viabiliza a utilização de técnicas e algoritmos, existentes e já consagrados, para a solução de problemas relacionados à mineração de textos, mais especificamente, à categorização de textos, utilizando como ambiente de execução *grids* computacionais.

Foram descritas as técnicas de classificação implementadas, detalhando os algoritmos bayesiano e de ranqueamento linear, procurando demonstrar seus fundamentos matemáticos, como também realizar um detalhamento sobre as métricas de avaliação utilizadas nesse tipo de processo.

As ferramentas utilizadas para a construção do portal assim como a abordagem sistêmica eleita para a realização da análise do sistema proposto, foram descritas e fundamentadas, de maneira a permitir que outros pesquisadores possam vir a agregar novos algoritmos ao sistema, com um mínimo de esforço.

O sistema Aîuri foi apresentado na *2nd EELA Grid School*, realizada na cidade de Mérida, na Venezuela, onde seus algoritmos foram testados em um ambiente de *grid* tradicional, ou seja, sem a utilização de serviços *web* ou serviços *grid*. Esta experiência foi considerada importante, uma vez que os testes realizados até então, haviam sido feitos somente no *grid* NACAD, que é um ambiente controlado, sobre o qual o sistema foi desenvolvido desde a sua

concepção. A utilização do sistema Aîuri com sucesso em um ambiente de *grid* real, com interfaces bem definidas e sem nenhuma possibilidade de acesso ao ambiente de configuração por parte do desenvolvedor, demonstrou que a tecnologia usada para o seu desenvolvimento foi adequada.

Inicialmente, a proposta do presente trabalho era a criação de um portal para a integração de algoritmos de mineração de textos a somente um ambiente de *grid* (NACAD), porém após esta bem sucedida experiência, foram incorporadas as funcionalidades necessárias na aplicação para a integração com este novo ambiente computacional. Deve-se ressaltar, que o esforço de desenvolvimento para esta nova abordagem foi mínimo, uma vez que a aplicação foi projetada com a utilização *design patterns* que facilitam a sua manutenibilidade.

Pode-se concluir diante dos experimentos realizados, que os resultados esperados foram atingidos. Fica evidente a possibilidade de integração de um portal para a mineração de textos, integrado a ambientes de *grids* computacionais. Além disso, o sistema Aîuri executou os experimentos de maneira rápida e eficiente.

Um fator que deve ser ressaltado em relação à esta ferramenta, é que não só algoritmos para mineração de textos podem ser incorporados. Seu objetivo maior é ser um portal acadêmico que possibilite a inclusão de algoritmos dos mais diversos ramos do conhecimento.

Alguns desafios tiveram que ser contornados ao longo do desenvolvimento do trabalho, e são relatados a seguir para que sirvam de experiência na realização de novas pesquisas:

(i) Documentos em formato PDF não podem ser manipulados pela ferramenta, tendo o seu conteúdo que ser copiado para um arquivo ASCII, para a partir daí ser tratado;

(ii) Deve-se ressaltar a importância de utilizar uma *stoplist* adequada à natureza dos textos a serem minerados. Assim, cada tipo de área de atuação merece a especificação própria desta metalinguagem;

(iii) Como foi dito no capítulo 3, a configuração de um *grid* computacional ainda é uma tarefa árdua, sendo necessária a documentação de cada detalhe da instalação para que seja replicado em outra máquina.

6.1 Trabalhos Futuros

Vários trabalhos podem ser desenvolvidos a partir da construção do portal Aîuri. Outras áreas podem ser investigadas e novas técnicas e variações de algoritmos podem ser comparadas dentro do sistema.

Os algoritmos implementados foram testados com a utilização de dicionários globais, porém com pequenas modificações poderiam ser utilizados dicionários locais, apesar de ser um procedimento questionável em relação à qualidade dos modelos gerados.

Alguns desafios relativos à interface podem ser implementados. A construção de um *workflow* científico para a configuração dos *jobs* é uma alternativa a ser discutida.

A integração com bancos de dados é uma alternativa viável e que pode ser implementada, de maneira a possibilitar uma forma de armazenamento de dados. Uma outra possibilidade seria a implementação de armazenamento distribuído sem a utilização de SGBD, com abordagens semelhantes ao OceanStore e HDF5.

Alternativas de pré-processamento em dispositivos móveis podem ser adotadas como uma alternativa ao processamento local realizado no portal. Interfaces definidas para possibilitar a interoperabilidade entre as tecnologias envolvidas devem ser discutidas.

Um último e importante desafio, é a migração do ambiente atual para um *framework* de portal de *grid* computacional que encapsule algumas das funcionalidades já implementadas, liberando, desta maneira, o desenvolvedor para a implementação de novos requisitos com o intuito de agregar novas funcionalidades ao sistema proposto.

Apêndice A

Inclusão de Novos Algoritmos

No desenvolvimento do sistema Aîuri, houve a preocupação com os fatores escalabilidade e manutenibilidade, de maneira a permitir que o *software* evolua sem grande esforço de manutenção. Para isso utilizou-se alguns padrões de projeto, que segundo Gamma et al. [103], são descrições de objetos e classes comunicantes que são customizados para resolver um problema geral de um projeto num contexto particular.

A seguir são apresentados os passos para a inclusão de novos algoritmos na aplicação. Todos os procedimentos foram realizados na ferramenta IDE Eclipse 3.2.

(i) Criação do item de menu: deve-se alterar o arquivo "lateral.jsp" e realizar a inclusão de um item de menu. A tag *href* deve possuir uma string que será mapeada no arquivo *struts-config.xml*. A figura A.1 ilustra este procedimento.

```
<a href="gerarBayes.do" target="mainFrame"><font color="#FFFFFF"><fmt:message
key="link.principal.gerarBayes"/></font></a>
<br>
<br>
```

Figura A.1: Inclusão de um item de menu

(ii) Criação da interface *jsp*: deve-se criar a interface *jsp* que será chamada quando o usuário clicar no menu. Os campos devem ser implementados seguindo o padrão *struts* [19]. A figura A.2 ilustra este padrão.

```
<td><font size="2" face="Verdana,Arial"><B>User:</B></font></td>
<td><font size="2" face="Verdana,Arial"> <html:text property="usuario"/></font></td>
```

Figura A.2: Padrão de campos na interface

(iii) Criação da classe *form* correspondente: deve-se criar uma classe no pacote *form* com os mesmos atributos (nomes idênticos) da interface *jsp*. Os métodos *get* e *set* devem ser implementados.

(iv) Criação da classe *DTO*: deve-se criar uma classe no pacote DTO com a mesma estrutura da classe *form*.

(v) Criação da classe de negócio: deve-se criar uma classe no pacote *modelo* que contenha todas as operações de negócio. Esta classe deve estender *AlgoritmoModelo*. A classe *AlgoritmoModelo* possui métodos abstratos que devem obrigatoriamente ser implementados na nova classe criada.

(vi) Inclusão do nome do novo algoritmo no *factory method*: na classe *AlgoritmoFactory* deve-se incluir uma nova opção com o nome do novo algoritmo. Esta classe implementa o padrão *Factory Method* [103]. A figura A.3 ilustra este procedimento.

```
package br.com.nacad.struts.modelo;

public class CategorizadorFactory
{
    public static CategorizadorModelo getCategorizador( String tipoCategorizador )
    {
        if( tipoCategorizador == null ) return null;
        else if( tipoCategorizador.equals("bayes") ) return new BayesModelo();
        else if( tipoCategorizador.equals("linear") ) return new LinearModelo();
        else return null;
    }
}
```

Figura A.3: Padrão Factory Method

(vii) Criação do *web service*: deve-se criar uma classe que implemente a *interface WebServiceInterface*. Esta nova classe deverá implementar o método *webService* para o novo algoritmo.

(viii) Inclusão do nome do novo algoritmo no *factory method*: na classe *WebServiceFactory* deve-se uma nova opção com o nome do novo algoritmo, assim como no item 6. Esta classe também implementa o padrão *Factory Method*.

(ix) Criação do *action*: no pacote *action* é necessária a criação de uma classe que estende *Action* e encapsula todos os procedimentos que serão realizados a partir de uma submissão. Esta *action* implementa o padrão *Command* [103].

(x) Mapeamento do *form* e da *action*: a classe *form* criada, assim como a *action* definida devem ser mapeadas no arquivo *struts-config.xml*. A figura A.4 ilustra este procedimento.

```
<struts-config>
    <!-- Form Bean Definitions -->
    <form-beans>
       <form-bean name="loginForm" type="br.com.nacad.struts.form.LoginForm"/>
       <form-bean name="converterXML1Form" type="br.com.nacad.struts.form.Conve
       <form-bean name="converterXML2Form" type="br.com.nacad.struts.form.Conve
       <form-bean name="converterXMLTeste1Form" type="br.com.nacad.struts.form
       <form-bean name="converterXMLTeste2Form" type="br.com.nacad.struts.form
       <form-bean name="testeGrid1Form" type="br.com.nacad.struts.form.TesteGr
       <form-bean name="gerarBayes1Form" type="br.com.nacad.struts.form.GerarBa
       <form-bean name="gerarLinear1Form" type="br.com.nacad.struts.form.Gerarl
       <form-bean name="stemmer1Form" type="br.com.nacad.struts.form.Stemmer1Fo
       <form-bean name="stemmer2Form" type="br.com.nacad.struts.form.Stemmer2Fo
    </form-beans>

    <global-forwards>
        <forward name="logon" path="/logon.do"/>
    </global-forwards>

    <!--  Action Mapping Definitions  -->
    <action-mappings>
        <action
            path="/login"
            type="br.com.nacad.struts.action.LoginAction"
            name="loginForm"
            scope="request"
            validate="false"
            input="/loginForm.jsp">
            <forward name="success" path="/principal.jsp"/>
            <forward name="err" path="/erro.jsp"/>
        </action>
```

Figura A.4: Arquivo struts-config.xml

(xi) Criação e publicação do *webService*: caso o algoritmo esteja no padrão do *globus 4*, o método *webService* da classe criada no pacote *modelo* deve ser publicado como um *web service*. Para isso, na IDE Eclipse deve-se selecionar a classe criada e dar um clique com o botão direito. Deve-se selecionar o menu *Web Services* e clicar na opção *Create Web Service*. A partir daí deve-se seguir as instruções da IDE.

Referências

[1] FELDMAN, R., DAGAN, I., "KDT - knowledge discovery in texts". In: *Proceedings of the First Int. Conf. on Knowledge Discovery (KDD)*, pp. 112–117, 1995.

[2] HOTHO, A., NURNBERGER, A., PAAB, G., "A brief survey of text mining". *LDV Forum* v. 20, n. 1, pp. 19–62, 2005.

[3] FAN, W., WALLACE, L., RICH, S., ZHANG, Z., "Tapping the power of text mining". *Communications of the ACM* v. 49, n. 9, pp. 77–82, Sep 2006.

[4] CHEN, M. S., HAN, J., YU, P. S., "Data mining: an overview from a database perspective". *IEEE Transaction on Knowledge and Data Engineering* v. 8, n. 6, pp. 866–883, 1996.

[5] MITCHELL, T., *Machine Learning*. New York, McGraw-Hill, 1997.

[6] HASTIE, T., TIBSHIRANI, R., FRIEDMAN, J., *The Elements of Statistical Learning*. Berlim, Springer-Verlag, 2001.

[7] B., M., HAND, D. J., *Intelligent data analysis*. New York, Springer-Verlag, 1999.

[8] MAITRA, R., "A statistical perspective on data mining". *Journal of the Indian Society for Probability and Statistics* v. 6, pp. 28–77, 2002.

[9] LANCASTER, F. W., *Indexação e resumos: teoria e prática*. Brasília, Briquet de Lemos/Livros, 1993.

[10] SEBASTIANI, F., "Machine learning in automated text categorization". *ACM Computing Surveys* v. 34, n. 1, pp. 1–47, Mar 2002.

[11] HIDALGO, J. M. G., "Text filtering at poesia: A new internet content filtering tool for educational environments". In: *Proceedings of SAC-02, 17th ACM Sysposium on Applied Computing*, pp. 615–620, 2002.

[12] HIDALGO, J. M. G., SANZ, E. P., RODRÍGUEZ, M. B., GARCÍA, F. C., "Text filtering at POESIA: A new internet content filtering tool for educational environments". *Procesamiento del Lenguaje Natural* v. 29, pp. 291–292, Jul 2002.

[13] DEBOLE, F., SEBASTIANI, F., "Supervised term weighting for automated text categorization". In: *Proceedings of SAC-03, 18th ACM Symposium on Applied Computing*, pp. 784–788, 2003.

[14] LEWIS, D. D., YANG, Y., ROSE, T., LI, F., "RCV1: A new benchmark colection for text categorization research". *Journal of Machine Learning Research* v. 5, pp. 361–397, Apr 2004.

[15] SOUCY, P., MINEAU, G. W., "Beyond TF-IDF weighting for text categorization in the vector space model". In: *Proceedings of IJCAI-05, 9th International Joint Conference on Artificial Intelligence*, pp. 1130–1135, 2005.

[16] HIDALGO, J. M. G. Text representation for automatic text categorization, 2003. URL `http://www.esi.uem.es/ jmgomez/tutorials/eacl03/index.html`. Conference of the European Chapter of the Assotiation for Computational Linguistics (EACL-2003), Madrid, ES.

[17] FOSTER, I., KESSELMAN, C., "The Grid: Blueprint for a Future Computing Infrastructure". chapter 2 of Computational Grids. Morgan Kaufmann Publishers, 1998.

[18] LARMAN, C., *Utilizando UML e Padrões*. Porto Alegre, Bookman, 2004.

[19] HUSTED, T., ET AL., *Struts em Ação*. Rio de Janeiro, Editora Ciência Moderna, 2004.

[20] HEARST, M., "Untangling text data mining". In: *Proceedings of ACL'99: the 37th Annual Meeting of the Association for Computational Linguistics*, pp. 20–26, 1999.

[21] MOONEY, R. J., NAHM, U. Y., "Text mining with information extraction". In: *Proceedings of the 4th International MIDP Colloquium*, pp. 141–160. Van Schaik, Sep 2003.

[22] DORRE, J., GERSTL, P., SEIKERT, R., "Text mining: Finding nuggets in mountains of textual data". In: *Knowledge Discovery and Data Mining*, pp. 398–401, San Diego, 1999.

[23] TAN, A., "Text mining: The state of the art and the challenges". In: *Proceedings of the Pacifffic Asia Conf on Knowledge Discovery and Data Mining PAKDD'99 workshop on Knowledge Discovery from Advanced Databases*, pp. 65–70, 1999.

[24] KROEZE, J. H., MATTHEE, M. C., BOTHMA, T. J. D., "Differentiating data and text mining terminology". In: *SAICSIT '03: Proceedings of the 2003 annual research conference of the South African institute of computer scientists and information technologists on Enablement through technology*, pp. 93–101, Republic of South Africa, 2003. South African Institute for Computer Scientists and Information Technologists. ISBN 1-58113-774-5.

[25] GRISHMAN, R., "Information extraction: Techniques and challenges". In: *SCIE*, pp. 10–27, 1997.

[26] MATSUNAGA, L. A., *Uma Metologia de Categorização de Textos para a Distribuição dos Projetos de Lei às Comissões Permanetes da Câmara Legislativa do Distrito Federal*. Ph.D. dissertation, Universidade Federal do Rio de Janeiro, Rio de Janeiro, Jun 2007.

[27] EBECKEN, N. F. F., LOPES, M. C. S., COSTA, M. C. A., "Sistemas Inteligentes: Fundamentos e Aplicações". chapter Mineração de Textos. São Paulo, Manole, 2003.

[28] CORDEIRO, A. D., *Gerador Inteligente de Sistemas com Auto-Aprendizagem para Gestão de Informações e Conhecimento*. Ph.D. dissertation, Universidade Federal de Santa Catarina, Santa Catarina, 2005.

[29] DA SILVA, C. F., *Uso de informações Linguísticas na etapa de pré-processamento em Mineração de Textos*. Master's Thesis, Universidade do Vale do Rio dos Sinos, São Leopoldo, Feb 2004.

[30] FAYYAD, U. M., PIATETSKY-SHAPIRO, G., SMYTH, P., "Knowledge discovery and data mining: Towards a unifying framework". In: *KDD*, pp. 82–88, 1996.

[31] KORFHAGE, R. R., *Information Retrievel and Storage*. New York, Jonh Wiley Sons, 1997. 349p.

[32] KOWALSKI, G., *Information Retrievel Systems: Theory and Implementation*. Boston, Kluwer Academic Publishers, 1997. 282p.

[33] SALTON, G., MACGILL, M., *Introduction to Modern Information Retrieval*. New York, McGRAW-Hill, 1983. 448p.

[34] PASSARIN, D. Text mining no aperfeiçoamento de consultas e definição de contextos de uma central de notícias baseada em rss, 2005. Palmas.

[35] BAEZA-YATES, R., "FRAKES,William B.; BAEZA-Yates, Ricardo A. Information Retrieval: Data Structures & Algorithms". chapter String Searching Algorithms, pp. 219–240. Upper Saddle River, New Jersey, Prentice Hall, 1992.

[36] FOX, C., "FRAKES,William B.; BAEZA-Yates, Ricardo A. Information Retrieval: Data Structures & Algorithms". chapter Lexical analysis and stoplists, pp. 102–130. Upper Saddle River, New Jersey, Prentice Hall, 1992.

[37] LEWIS, D. D., "Evaluating text categorization". In: *Speech and Natural Language Workshop*, pp. 312–318. Morgan Kaufmann, 1991.

[38] ORENGO, V. M., HUYCK, C. R., "A stemming algorithim for the portuguese language". In: *Proceedings of the SPIRE Conference. Laguna de San Raphael: [s.n.]*, pp. 13–15, 2001.

[39] PORTER, M. F., "An algorithm for suffix stripping". *Program: electronic library and information systems* v. 14, n. 3, pp. 130–137, 1980.

[40] LOVINS, J. B., "Development of a stemming algorithm". *Mechanical Translation and Computational Linguistics 11* v. 1, n. 2, pp. 22–31, 1968.

[41] PAICE, C., "An evaluation method for stemming algorithms". In: *Proceedings Conference on Research and Development in Information Retrieval*, pp. 42–50, London, 1994. Springer Verlag.

[42] HARMAN, D., "How effective is suffixing". *Journal of the American Society for Information Science* v. 42, n. 1, pp. 7–15, 1991.

[43] LENNON, M., PIERCE, D., TARRY, B., WILLET, P., "An evaluation of some conflation algorithms for information retrieval". *Journal of Information Science* v. 3, pp. 177–183, 1981.

[44] SEBASTIANI, F., "A tutorial on automated text categorization". In: *In Analia Amandi and Alejandro Zunino (eds.) Proceedings of ASAI-99 1st Argentinian Symposium on Artificial Intelligence Buenos Aires AR*, pp. 7–35, 1999.

[45] MEADOW, C., BOYCE, B., KRAFT, D., *Text Information Retrieval Systems*. San Diego, Academic Press, 2000.

[46] RIJSBERGEN, C. V., *Information Retrieval*. 2 ed., London, Butterworths, 1979. 147p.

[47] SALTON, G., BUCKLEY, C., *Improving retrieval performance by relevance feedback*. Department of Computer Science-Cornell University, TECHNICAL REPORT, 1987.

[48] SILVA, C. F., *Uso de Informações Linguísticas na etapa de pré-processamento em Mineração de Textos*. Master's Thesis, Universidade do Vale do Rio dos Sinos, São Leopoldo, Feb 2004.

[49] BEKKERMAN, R., EL-YANIV, R., TISHBY, N., WINTER, Y., "Distributional word clusters vs words for text categorization". *Journal of Machine Learning* v. 3, pp. 1183–1208, 2003.

[50] BAEZA-YATES, R., RIBEIRO-NETO, B., *Modern Information Retrieval*. Addison-Wesley, 1999.

[51] RIORDAN, C., SORENSEN, H. Information filtering and retrieval: An overview. URL `http://citeseer.nj.nec.com/483228.html`.

[52] LOPES, M. C. S., *Mineração de Dados Textuais utilizando Técnicas de Clustering para o Idioma Português*. Ph.D. dissertation, COPPE/UFRJ, Rio de Janeiro, Oct 2004. Tese de Doutorado do Programa de Engenharia Civil da COPPE/UFRJ.

[53] WIVES, L. K., *Um estudo sobre técnicas de recuperação de informações com Ênfase em informações textuais*. Universidade Federal do Rio Grande do Sul, , Rio Grande do Sul, 1997. Programa de Pós-Graduação em Computação.

[54] WIVES, L. K., *Tecnologias de descoberta de conhecimento em textos aplicadas à inteligência competitiva*. Universidade Federal do Rio Grande do Sul, , Rio Grande do Sul, 2002. Programa de Pós-Graduação em Computação.

[55] YANG, Y., PEDERSON, J., "A comparative study on feature selection in text categorization". In: *Proceedings of 14th International Conference on Machine Learning*, pp. 412–420, San Francisco-US, 1997. Morgan Kaufmann.

[56] GUTHRIE, L., PUSTEJOVSKY, J., WILKS, Y., SLATOR, B. M., "The role of lexicons in natural language processing". *Communications of the ACM* v. 39, n. 1, pp. 63–72, 1996.

[57] JACOBS, P. S., RAU, L. F., "Innovations in text interpretation". *Artificial Intelligence* v. 63, n. 1-2, pp. 143–191, 1993.

[58] WIVES, L. K., LOH, S., "Tecnologias de descoberta de conhecimento em informações textuais; ênfase em agrupamento de informações". In: *OFICINA DE INTELIGÊNCIA ARTIFICIAL (OIA)*, pp. 28–48, Pelotas-RS-Brasil, 1999. EDUCAT.

[59] SPARK-JONES, K., WILLET, P., *Readings in Information Retrieval*. San Francisco, Morgan Kaufmann, 1997.

[60] JOACHIMS, T., *Learning to Classify Text Using Support Vector Machines*. Kluwer Academic Publishers, 2002. 230p.

[61] BASTOS, V. M., *AMBIENTE DE DESCOBERTA DE CONHECIMENTO NA WEB PARA A LÍNGUA PORTUGUESA*. Ph.D. dissertation, COPPE/UFRJ, Rio de Janeiro, Oct 2006. Tese de Doutorado do Programa de Engenharia Civil da COPPE/UFRJ.

[62] LANCASTER, F. W., *Information Retrieval Systems: Characteristics, Texting and Evaluation*. San Francisco, John Wiley Sons, 1968. 222p.

[63] WEISS, S. M., INDURKHYA, N., ZHANG, T., DAMERAU, F. J., *Text Mining: Predictive Methods for Analyzing Unstructured Information*. New York, Springer Science+Business Media, 2005. 237p.

[64] TZERAS, K., HARTMAN, S., "Automatic indexing based on bayesian inference networks". In: *Proceedings 16th ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR93)*, pp. 22–34, 1993.

[65] LEWIS, D. D., RINGUETTE, M., "Comparison of two learning algorithms for text categorization". In: *Proceedings of the Third Annual Symposium on Document Analysis and Information Retrieval (SDAIR94)*, 1994.

[66] LACERDA, W. S., BRAGA, A. P., "Experimento de um classificador de padrões baseado na regra naive de bayes". *INFOCOMP - Revista de Computação da UFLA - Lavras* v. 1, n. 3, pp. 30–35, 2004.

[67] LEWIS, D. D., "Naïve (bayes) at forty: The independence assumption in information retrieval". In: *Proceedings of ECML-98, 10th European Conference on Machine Learning*, pp. 4–15, Chemnitz-Germany, 1998.

[68] GEMERT, J. V., *Text mining tools on the internet*. University of Amsterdam, , Sep 2006. vol. 25.

[69] KONCHADY, M., *Text Mining Applications Programming*. Massachusetts, Charles River Media, 2006.

[70] TANG, Z. T., MACLENNAN, J., *Data Mining with SQ Server 2005*. Indiana, Wiley Publishing Inc., 2005.

[71] FOSTER, I., KESSELMAN, C., TUECKE, S., "The anatomy of the grid: Enabling scalable virtual organizations". *Lecture Notes in Computer Science* v. 2150, 2001.

[72] NASSIF, L. N., *Seleção distribuída de recursos em grades computacionais usando raciocínio baseado em casos e políticas de granularidade fina*. Ph.D. dissertation, UFMG, Jun 2006.

[73] DANTAS, M., *Computação Distribuída de Alto Desempenho, Redes, Clusters e Grids Computacionais*. Axcel Books do Brasil Editora, 2005. ISBN 85-7323-169-6.

[74] FOSTER, I., KELSSELMAN, C., *The Grid2: Blueprint for a New Computing Infrastructure*. San Francisco, Morgan Kaufmann Publishers, 2004.

[75] BERTIS, V. Fundamentals of grid computing, 2002. URL `http://www.redbooks.ibm.com/redpapers/pdfs/redp3613.pdf`.

[76] NASH, M. Oracle 10g: Infrastructure for grid computing, 2004. URL `http://otn.oracle.com/tech/grid/collateral/GridTechWhitePaperfinal.pdf`.

[77] MALAIKA, S., EISENBERG, A., MELTON, J., "Standards for databases on the grid". *Sigmod Record* v. 32, n. 3, pp. 92–100, 2003.

[78] FOSTER, I., KESSELMAN, C., NICK, J., TUECKE, S., "The physiology of the grid: An open grid services architecture for distributed systems integration. open grid service infrastructure wg". , 2001.

[79] FOSTER, I., "What is the grid? a three point checklist". *GRIDToday* v. 1, n. 6, , Jul 2002.

[80] SKILLCORN, D. B., "Motivating computational grids". In: *Proceedings of the 2nd IEEE/ACM Internacional Symposium on Cluster Computing and the Grid*, Mar 2002.

[81] PARRA, A., *Mecanismo de Matching Semântico de Recursos Computacionais de Grids baseado na Integração Semântica de Múltiplas Ontologias*. Master's Thesis, Universidade Federal de Santa Catarina, Santa Catarina, Sep 2006.

[82] SKILLCORN, D. B., *The case for datacentric grids*. Queen's University, , Nov 2001. Departament of Computing and Information Science.

[83] CHIN, J., COVENEY, P. V., "Towards tractable toolkits for the grid: a plea for lightweight, reusable middleware". , n. UKeS-2004-01, , 2005.

[84] SOTOMAYOR, B. The globus toolkit 3 programmer's tutorial, Dec 2003. URL `http://gdp.globus.org/gt3-tutorial`.

[85] CIRNE, W., SANTOS-NETO, E. Grids computacionais: Da computação de alto desempenho a serviços sob demanda. Mini-curso no 23o Simpósio Brasileiro de Redes de Computadores, 2005.

[86] Growing animated film talent. URL `http://www.hpl.hp.com/SE3D/se3d-overview.html`.

[87] Open grid services infrastructure (ogsi). URL `http://forge.gridforum.org/projects/ggf-editor/document/draftogsi-service-1/en/1`.

[88] CZAJKOWSKI, K., FERGUSON, D., FOSTER, I., FREY, J., GRAHAM, S., MAGUIRE, T., SNELLING, D., TUECKE, S., *From ogsi to ws-resource framework: Refactoring and evolution*. IBM, , 2004.

[89] FORUM, G. G. Open grid services architectur (ogsa). URL `http://forge.gridforum.org/projects/ggf-editor/ogsa`.

[90] SOTOMAYOR, B. Globus toolkit 4 programmer´s tutorial, 2005. URL `http://gdp.globus.org/gt4-tutorial/multiplehtml/index.html`.

[91] The web service description language - wsdl. URL `http://www.w3.org/TR/wsdl`.

[92] BANKS, T. Web services resource framework (wsrf) - primer v1.2. URL `http://docs.oasis-open.org/wsrf/wsrf-primer-1.2-primer-cd-02.pdf`.

[93] HUMPHREY, M., WASSON, G., MORGAN, M., BEEKWILDER, N., "An early evaluation of wsrf and ws-notification via wsrf.net". , pp. 172–181, Nov 2004.

[94] FOSTER, I., KESSELMAN, C., "Globus: A metacomputing infrastructure toolkit". *The International Journal of Supercomputer Applications and High Performance Computing* v. 11, n. 2, pp. 115–128, 1997.

[95] REYNOLDS, H., KOULOPOULOS, T., "Enterprise knowledge has a face". *Intelligent Enterprise* v. 2, n. 5, pp. 29–34, Mar 1999.

[96] Sun technical computing portal, . URL `http://sun.com/solutions/hpc/pdfs/TCP-final.pdf`.

[97] Java technology, . URL `http://java.sun.com`.

[98] Software composition. URL `http://wiki.compor.net`.

[99] Borland jbuilder. URL `http://www.borland.com/us/products/jbuilder/index.html`.

[100] Eclipse foundation. URL `http://www.eclipse.org`.

[101] CAMPIONE, M., WALRATH, K., *The Java Tutorial: Object-Oriented Programming for the Internet*. SunSoft Press, 1996.

[102] GOSLING, J., MCGILTON, H. The java language: A white paper, 1995.

[103] GAMMA, E., HELM, R., JOHNSON, R., VLISSIDES, J., *Padrões de Projeto - Soluções reutilizáveis de Software Orientado a Objetos*. Bookman, 2004.

[104] JOHNSON, R., FOOTE, B., "Designing reusable classes". *Journal of Object-Oriented Programming* v. 1, n. 5, pp. 22–35, Jun 1988.

[105] JACOBSON, I., BOOCH, G., RUMBAUGH, J., *The Unified Software Development Process*. Massachussetts, Addison-Wesley, 1999.

[106] BOOCH, G., RUMBAUGH, J., JACOBSON, I., *UML Guia do Usuário*. Rio de Janeiro, Campus, 2000.

[107] GLOBUS. Globus toolkit. URL `http://www.globus.org/toolkit/docs/4.0/admin/`

[108] KOBLITZ, B., SANTOS, N., "*AMGA User's and Administrator's Manual*", Nov 2006.

[109] SCIFO, S., "Gsaf - grid storage access framework". , Jun 2007.

[110] PACINI, F. Job description language - howto, Dec 2001.

[111] Cetenfolha - conjunto de textos para mineração de textos. URL `http://http://acdc.linguateca.pt/cetenfolha/`.