



Pós-Graduação em Ciência da Computação

“Applying a Semantic Layer in a Source Code Retrieval Tool”

Por

Frederico Araujo Durão

Dissertação de Mestrado



Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE, ABRIL/2008



Universidade Federal de Pernambuco

CENTRO DE INFORMÁTICA

PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Frederico Araujo Durão

"APPLYING A SEMANTIC LAYER IN A SOURCE CODE RETRIEVAL TOOL"

ESTE TRABALHO FOI APRESENTADO À PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO DO CENTRO DE INFORMÁTICA DA UNIVERSIDADE FEDERAL DE PERNAMBUCO COMO REQUISITO PARCIAL PARA OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIA DA COMPUTAÇÃO.

ORIENTADOR: Silvio Romero de Lemos Meira

CO-ORIENTADOR: Eduardo Santana de Almeida

RECIFE, ABRIL/2008

Durão, Frederico Araujo

**Applying a semantic layer in a source code
retrieval tool / Frederico Araújo Durão. – Recife: O
Autor, 2008.**

127 folhas : il., fig., tab.

**Dissertação (mestrado) – Universidade Federal de
Pernambuco. Cln. Ciência da Computação, 2008.**

Inclui bibliografia e apêndice.

1. Ferramenta de busca. I. Título.

025.524

CDD (22.ed.)

MEI2008-033



Table of Contents

Acknowledgments	4
Resumo	5
Abstract.....	6
List of Acronyms.....	7
List of Figures.....	8
List of Tables	9
Chapter 1 - Introduction	10
1.1. Motivation.....	10
1.1.1. Source Code Search Engines	11
1.2. Problem Statement	12
1.3. Overview of the Proposed Solution	12
1.3.1. Context	12
1.3.2. Outline of the Proposal	13
1.4. Out of Scope.....	14
1.5. Statements of the Contribution	15
1.6. Organization of the Dissertation	16
Chapter 2 - The Software Reuse: An Overview.....	18
2.1. Introduction.....	18
2.2. The Motivation and Benefits of Software Reuse	20
2.3. Software Reuse Impediments.....	21
2.4. Reusable Software Assets	23
2.4.1. Source Code Reuse.....	24
2.4.2. Accessing Reusable Software Assets	27
2.5. Chapter Summary.....	27
Chapter 3 - Information Retrieval.....	29
3.1. Introduction.....	29
3.2. Information Retrieval versus Data Retrieval.....	30

3.3.	The Relevant Information	32
3.4.	The Semantic Challenges of Information Retrieval Systems	33
3.5.	Technical Support for Information Retrieval Systems	34
3.5.1	Techniques for Query Formulation Support.....	35
3.5.2	Classification Schemes for Component Retrieval Support	36
3.6.	Source Code Retrieval for Reuse Activity	38
3.7.	Information Retrieval Systems for Component Reuse.....	39
3.8.	Information Retrieval Discussion.....	42
3.9.	Chapter Summary	43
Chapter 4 - The Semantic Web		44
4.1.	Introduction.....	44
4.2.	Ontology.....	45
4.3.	Real-world Ontology Aspects.....	46
4.4.	Ontology Engineering	47
4.5.	Ontology Levels.....	49
4.6.	The Mark-up Ontology Web Languages	50
4.6.1.	Resource Description Framework (RDF)	51
4.6.2.	RDF Schema.....	52
4.6.3.	Ontology Inference Layer (OIL).....	52
4.6.4.	Ontology Web Language (OWL)	53
4.6.5.	DARPA Agent Markup Language (DAML)	54
4.7.	Outstanding Semantic Web Projects for Knowledge Management	54
4.8.	Semantic Search Engines.....	59
4.8.1	Semantic Search Engines	61
4.9.	Chapter Summary	62
Chapter 5 - Semantic Search Engine		63
5.1.	Requirements.....	64
5.1.1.	Functional Requirements.....	64
5.1.2.	Non-Functional Requirements	66
5.2.	System Architecture.....	70
5.2.1.	B.A.R.T Client.....	70
5.2.2.	B.A.R.T Server	71
5.3.	The Semantic Components.....	73
5.3.1.	Semantic Code Analyzer.....	73
5.3.1.1.	Implementation Aspects.....	75
5.3.1.2.	Source Code Analysis and Classification	76

5.3.1.3.	Semantic Indexing.....	78
5.3.2.	Semantic Query Reasoner.....	79
5.3.2.1.	Implementation Aspects.....	81
5.3.2.2.	The Ontology Model.....	82
5.4.	Semantic Configuration at B.A.R.T Eclipse Plug-in	86
5.5.	Reused Frameworks	87
5.6.	Semantic Search In Action.....	89
5.7.	Chapter Summary.....	93
Chapter 6 - Semantic Search Engine Evaluation.....		94
6.1.	Methodology	94
6.1.1.	Information Retrieval Evaluation.....	95
6.1.2.	Evaluation of the Semantic Layer	96
6.1.3.	Goal	96
6.1.4.	Variables.....	97
6.2.	Experiment Configuration and Instantiation.....	98
6.3.	The Evaluation Results	102
6.3.1.	Recall Results	102
6.3.2.	Precision Results.....	103
6.3.3.	F-Measure Results	104
6.4.	Final Considerations and Discussion	105
6.5.	Chapter Summary.....	109
Chapter 7 - Conclusion.....		110
7.1.	Achieved Goals.....	110
7.2.	Related Work and Research Foundations	111
7.2.1.	Software Reuse.....	112
7.2.2.	Information Retrieval	112
7.2.3.	Semantic Web	114
7.3.	Future Work.....	115
7.4.	Concluding Remarks.....	117
References.....		118
Appendix		126



cknowledgments

I would like to thank everyone that in some way have contributed and guided me to accomplish this important mission in my life.

There were so many people involved that I would not possibly be able to name them all here, so please accept my sincere apologies and be sure that I will always be grateful to all of you.

I certainly would not be able to complete this dissertation without entire emotional support from my family and close friends, especially my parents and brothers, outlining here, my grandmother Dona Maria Helena, for being a proof of love in my life, my lovely girl Renata Matos for the constant support and “patient” during the whole time of this research.

The RiSE staff, especially Alexandre Martins for the memorable technical discussions, of course, my co-adviser and always friend Eduardo Almeida, the reuse man who have found me in Bahia and my visionary and “out of the ordinary” adviser Silvio Meira, that have also been of fundamental importance with all the bright ideas and suggestions, guidance, valuable feedback and hard work. Silvio seems to be a water mine that everyday get renewed of life and youth.

Last but not least, I would like to C.E.S.A.R. software factory for providing all infrastructure for my personal growth and basis of this research.



esumo

O reuso de software é uma área de pesquisa da engenharia de software que tem por objetivo prover melhorias na produtividade e qualidade da aplicação através da redução do esforço. Trata-se de reutilizar artefatos existentes, ao invés de construí-los do zero a fim de criar novas aplicações. Porém, para obter os benefícios inerentes ao reuso, alguns obstáculos devem ser superados como, por exemplo, a questão da busca e recuperação de componentes. Em geral, há uma lacuna entre a formulação do problema, na mente do desenvolvedor e a recuperação do mesmo no repositório, o que resulta em resultados irrelevantes diminuindo as chances de reuso. Dessa maneira, mecanismos que auxiliem na formulação das consultas e que contribuam para uma recuperação mais próxima à necessidade do desenvolvedor, são bastante oportunos para solucionar os problemas apresentados.

Nesse contexto, este trabalho propõe a extensão de uma ferramenta de busca por palavra-chave através de uma camada semântica que tem por objetivo principal aumentar a precisão da busca e, conseqüentemente, aumentar as chances de reuso do componente procurado. A criação da camada semântica é representada basicamente por dois componentes principais: um para auxiliar o usuário na formulação da consulta, através do uso de uma ontologia de domínio, e outro para tornar a recuperação mais eficiente, através de uma indexação semântica dos componentes no repositório.

Palavras-chave: Semântica, Reuso de Software, Engenheiros de Busca, Ontologias



Abstract

The challenge for achieving a more efficient software engineering practice comprises a vast number of obstacles related to the intrinsic complexity of software systems and their surrounding contexts. Consequently, software systems tend to fail in meeting the real needs they were developed to address, consuming more resources, thus having a higher cost, and taking longer to complete than anticipated. The software reuse field is often regarded as the most promising discipline for closing these gaps, however, models and tools are still immature to make its adoption on a systematic fashion.

To promote the development of practices, models and tools are welcome activities to boost the reuse activity in most software development organizations. The lack of knowledge about reusable assets and the use of inappropriate tools are example of reasons for the low reuse activity. In this sense, this work presents a *semantic layer applied to a source code search tool* with the objective of bringing real relevant returns closer to user need, and, consequently to increase the chance of reuse. Two new components are proposed for the execution of the semantic activities and the resulting semantic search engine is evaluated with a realistic environment configuration analogous to projects from software organizations.

Keywords: Software Reuse, Information Retrieval, Semantic Web, Ontologies



ist of Acronyms

API : Application Programming Language

B.A.R.T :Basic Asset Retrieval Tool

CBD :Component Based-Development

CVS :Concurrent Version System

DAML :DARPA Agent Markup Language

GUI :Graphical User Interface

HTML :Hypertext Markup Language

IR :Information Retrieval

J2EE :Java Enterprise Edition

J2SE :Java Standard Edition

OIL :Ontology Inference Layer

OWL :Ontology Web Language

RDF :Resource Description Framework

RiSE :Reuse in Software Engineering

SVN :Support Vector Machine

SWT :Standard Widget Toolkit

URI : Uniform Resource Identifier

W3C :World Wide Web Consortium

WWW :World Wide Web

XML :Extensible Markup Language



ist of Figures

Figure 1.1. The RiSE framework.....	13
Figure 4.1. RDF diagram	51
Figure 4.2. Protégé Screenshot.	56
Figure 5.1. B.A.R.T. Architecture	71
Figure 5.2. Component communication	75
Figure 5.3. Source Code Classification	76
Figure 5.4. A feasible Java code for database domain.....	77
Figure 5.5. The metadata domain in the index structure	78
Figure 5.6. Semantic Code Analyze Viewer.....	79
Figure 5.7. Semantic Possibilities at Eclipse Plug-in.....	80
Figure 5.8. Component communication.....	81
Figure 5.9. Screenshot of the Ontology Model	83
Figure 5.10. Screenshot of the OWL document.....	84
Figure 5.11. Term View tab.....	85
Figure 5.12. B.A.R.T Configuration Screen.....	86
Figure 5.13. Query Reformulation.....	89
Figure 5.14. Screenshot of the B.A.R.T Eclipse plug-in	90
Figure 5.15. Query Expansion	91
Figure 5.16. Retrieval of Semantic Search	92
Figure 5.17. Final Search Results at B.A.R.T Search Plug-in	92
Figure 6.1. The evolution of B.A.R.T search mechanisms	106



ist of Tables

Table 3.1. Information Retrieval x Data Retrieval	31
Table 5.1. Summary of requirements	69
Table 6.1. Source Code Information	99
Table 6.2. Achieved Recall Rates	102
Table 6.3. Achieved Precision Rates	103
Table 6.4. Achieved F-Measure Rates	104
Table 6.5. Achieved Metrics	106
Table A.1. Queries utilized in the evaluation	126



Introduction

A fundamental principle for reusing software assets is to provide ways for accessing them. In this sense, information retrieval mechanisms combined with semantic web technologies play a very important role in finding relevant information. *Software Reuse*, *Information Retrieval* and the *Semantic Web* are the main subjects of this work and their foundations were studied and applied in the creation of a *semantic search engine* to promote the reuse activity by reducing the efforts during the software development.

This chapter contextualizes the focus of this dissertation and starts by presenting its motivation in Section 1.1 and a clear definition of the problem in Section 1.2. A brief overview of the proposed solution is presented in Section 1.3 while Section 1.4 describes some related aspects that are not directly addressed by this work. Section 1.5 presents the main contributions and, finally, Section 1.6 outlines the structure of the remainder of this dissertation.

1.1. Motivation

The dissemination of the *Information Technology* in the modern society jointly with the advances in *Computer Science* has driven many organizations of most different areas to adopt computational systems in order to become more competitive and survive in the wild trade market. However, such systems started to become more complex and this requires more qualified people constantly updated with the new trends. In this scenario, it is not difficult to imagine how similar applications have been developed independently, in different parts of the world and in different times in history, without sharing or reuse previously knowledge.

In this context, *Software Reuse* has emerged to skirt the “*redundant development*”, on the premise that, if accurately applied, assets previously built might be **reused instead of developed new ones from the scratch**. Moreover, reusing software assets might speed up the time-to-market with the improvement of the productivity and reduce the costs of the final product without decrease the quality. *Software Reuse*, in all its variances, is generally regarded as one of the most important mechanisms for performing software development more efficiently [McIlroy, 1968]. This belief has been systematically enforced by empirical studies that have, over the years, demonstrated the reuse effects on the software development in terms of quality, time-to-market and costs [Devanbu et al., 1991] [Lim, 1994] [Basili et al., 1996] [Frakes and Succi, 2001].

Nevertheless, in order to achieve such benefits, the adoption of a systematic reuse program which includes, for example, investments in tools to promote the reuse activity such as **source code search tools** remains necessary [Rine, 1997]. Such tools allow software developers to efficiently search, retrieve and reuse source code from many different repositories avoiding the writing of brand new codes since the same solution may have been implemented by somebody else [Morisio et al., 2002].

In spite of knowing that other tools such as reuse repositories [Burégio, 2006], tools for software reengineering [Brito, 2007], and systems for domain analysis [Lisboa et al., 2007] offer some of the mentioned reuse benefits, this dissertation concentrates its efforts on information retrieval tools dedicated to source code retrieval.

1.1.1. Source Code Search Engines

Source code search engines can be seen as effective tools for promoting source code reuse due to the amount of code spread over legacy systems in the organizations or distributed in the repositories on the Internet. Keyword-based search engines are example of tools employed for source code retrieval, nevertheless, the effectiveness of the theses source code search engines depend on the relevance of the query results. Therefore, efforts in the search quality can be considered an essential requirement for their reputation and popularity.

Based on these aspects, information retrieval techniques are useful in order to provide means of somehow interpreting its contents in an intelligent way, and retrieve the most relevant ones to the final user.

1.2. Problem Statement

In spite of contributing for reuse activity, keyword-based search engines utilized in software development face the problem of low relevance on its asset returns due the fact that a single keyword may not represent the desired functionality. This happens because no code content is analyzed or the query utilized does not represent clearly the user necessity [Ye and Fischer, 2002].

In a nutshell, traditional keyword-based search engines provide a user interface to specify criteria about an item of interest and look for the occurrence of the string pattern (or query) in source codes. There is no “query support” to assist users to express their needs by means of natural language processing or knowledge exploration about the repository content. For this reason, **the problem of locating relevant reusable source codes configures a key point for an effective source code reuse**. Therefore, semantic mechanisms to enhance traditional keyword-base search engines are highly advisable in order to improve the search efficiency as well as the software reuse.

1.3. Overview of the Proposed Solution

In order to accomplish the goal of this dissertation, a *Semantic Layer Applied to a Source Code Search Engine* is proposed. This Section presents the context where it is regarded and outlines the proposed solution.

1.3.1. Context

This work is part of a broader reuse initiative promoted by the *Reuse in Software Engineering* (RiSE)¹ research group [Almeida et al., 2004].

¹ <http://www.rise.com.br>

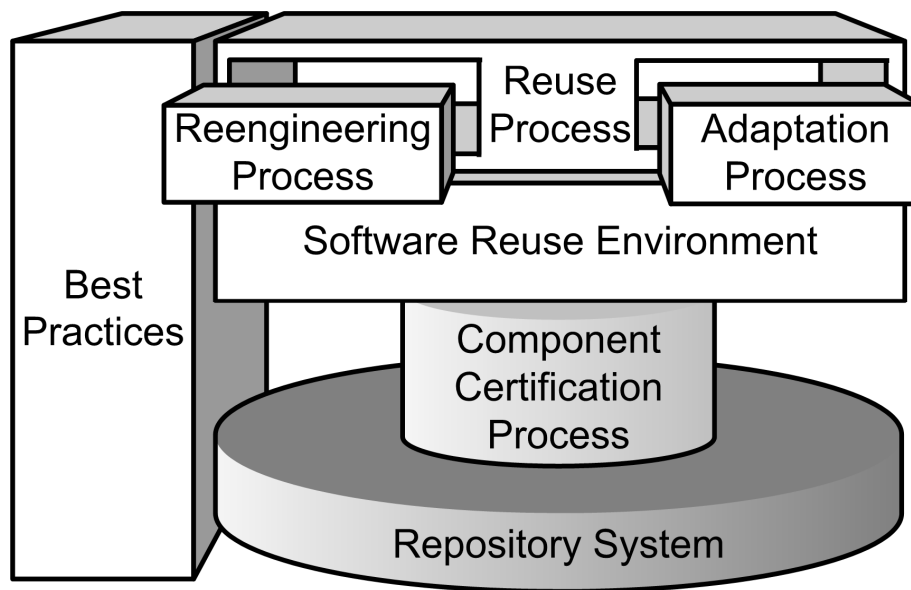


Figure 1.1. The RiSE framework

Figure 1.1 shows the RiSE framework which embraces a wide range of studies in areas such as reuse processes [Almeida et al., 2005a], domain engineering [Almeida, 2007], component certification [Alvaro et al., 2006] and repository system [Burégio, 2006]. Outstanding tools also are part of the RiSE Software Reuse environment such as the Maracatu search engine [Garcia et al., 2006b] which was enhanced with the facet-based and folksonomy mechanisms [Vanderlei et al., 2007]; the Admire Environment [Mascena, 2006], the Basic Asset Retrieval Tool (B.A.R.T) [Santos et al., 2006] and ToolDay, a Domain Analysis Tool [Lisboa et al., 2007]. These efforts are coordinated and will be integrated in a full-fledged enterprise scale reuse solution. The role of the *semantic layer* on the RiSE project is to provide improvements in the source code retrieval of B.A.R.T search engine with the objective of improving the search precision and as a consequence increase the chance of code reuse.

1.3.2. Outline of the Proposal

Aware of the problem of locating relevant source code, this dissertation presents an extended version of the B.A.R.T search engine enhanced with a semantic layer in order to improve the search precision and consequently the relevance of

the codes returned. According to [Clarke and Willett, 1997], the search precision provides an indication of the system relevance: the higher a search precision is, the higher its relevance is.

The proposed semantic layer consists of two core components that work in conjunction with existing architecture to provide the required semantic features. The objective, therefore, is **to combine the original keyword-based search mechanism with the semantic features** represented by ontology reasoning for assistance in query construction and machine learning technique for code comprehension for efficient search retrieval.

1.4. Out of Scope

Since the proposed solution aims to raise evidences about the benefits acquired with semantic layer, a set of related aspects will be left out of its scope. Even though the provided functionalities are based on well-founded aspects of quality and performance, they do not discard future enhancements to answer more efficiently its purpose. The aspects not directly addressed by this work are listed in the following:

- **Semantic Search of other Software Asset Types** – Due to the focus of this dissertation, the semantic layer was centered on *source code* retrieval; however, other asset types such as requirement documents, design architectural models and test plans may be considered in future versions;
- **Source Code Retrieval** – One of the most important requirement of the proposed solution is extensibility and, although any programming language may be included to the environment, the initial implementation only contemplates *Java* source files;
- **Supporting Functionalities** – Some of the functionalities proposed in the entire solution, such as the *Automatic Knowledge Base Population* and *Optimization of Graphical Semantic Classification* were not be implemented in this initial version. These functionalities, although important, consist on supporting operations for the core functionalities and thus may be incrementally implemented in future versions;

- **Non Use of Formal Semantics** - In the entire work, the statement “*semantic of source code*” will be constantly mentioned, however, this does not refer to formal semantics in which computer programs are represented through mathematical functions [Nielson and Nielson, 1992]. Although the use of formal semantics are not discarded in future releases, the semantics treated in this dissertation are in a higher level of abstraction and correspond to the functionalities performed by the source codes. A more detailed explanation about this is given in the Chapter 5 which describes the entire implementation aspects.

In a general view, although this initial implementation corresponds to the academic version and presents some limitations, the entire development primed for important attributes of quality such as modularization, readability and reusability by the fact that the results can be applied in a commercial version the of B.A.R.T search engine.

1.5. Statements of the Contribution

As a result of the work presented in this dissertation, a list of contributions may be enumerated:

- Creation of components to assist a keyword-based search engine to recover source code more efficiently;
- Proposition of an ontology model comprising a set of technical terms and infrastructure domains handled by source codes;
- Evolution of previous B.A.R.T architecture through the integration of the new (semantic) modules with low coupling without decreasing the system; and
- Implementation and evaluation of a reuse semantic layer based on the requirements that prime for quality and performance of the search engine. The provided implementation also is flexible enough to be reused in other keyword-based search engines.

Besides the final contributions listed so far, some intermediate results of this work have been reported in the literature, as shown in the following:

- Garcia, V. C., Lucrédio, D., Durão, F. A., Santos, E. C. R., Almeida, E. S., Fortes, R. P. M., Meira, S. R. L., *From Specification to Experimentation: A Software Component Search Engine Architecture*. The 9th International Symposium on Component-Based Software Engineering (CBSE 2006), Västerås, Sweden, Springer-Verlag, 2006;
- Vanderlei, T. A., Durão, F. A., Martins, A. C., Almeida, E. S., Meira, S. R. L., *A Classification Mechanism for Search and Retrieval Software Components*. 22nd Annual ACM Symposium on Applied Computing (SAC), Information Retrieval Track, Seul, Korea, 2007; and
- Durão, F. A., Vanderlei, T. A., Almeida, E. S., Meira, S. R. L. *Applying a Semantic Layer in a Source Code Search Tool*, 23rd Annual ACM Symposium on Applied Computing (SAC), Information Retrieval Track, Fortaleza, Ceará, Brazil, 2008. (To Appear).

1.6. Organization of the Dissertation

The remainder of this dissertation is organized as follows:

- Chapter 2 contains a comprehensive revision of the *Software Reuse* field with the goal of presenting the main concepts and identifying the major impediments for its diffusion besides some solutions adopted in previous studies;
- Chapter 3 reviews the *Information Retrieval* field with an emphasis to the instantiation of the information retrieval techniques in the software reuse;
- Chapter 4 presents an overview of *Semantic Web* field listing its purpose, ongoing projects, methodologies of application besides the benefits acquired with the advent of the *Semantic Web Markup Languages*;
- Chapter 5 describes the proposed semantic layer applied to the B.A.R.T search engine in details. The requirements, used technologies, architecture and the whole implementation aspects of the solution are discussed;

- Chapter 6 reports the entire environment used for the execution of the semantic search evaluation. In addition to the experiment environment, this Chapter also presents the expected goals to be achieved, the methodology adopted, the formal hypotheses analyzed, the final results and a discussion about the findings and the problems found; and
- Chapter 7 concludes this dissertation by summarizing the findings of the work and comparing it with some related studies. Future enhancements to the proposed solution are also discussed and some concluding remarks are presented.



The Software Reuse: An Overview

Considering that the main motivation of this work is to improve the reuse activity, an appropriate revision of the Software Reuse concepts is necessary to establish the foundations of this approach. This Chapter presents the key points of the Software Reuse field and discusses the reasons why, in spite of its promises, it is still not a widespread practice. The discussion is organized as follows: Section 2.1 presents an introduction to the software reuse field from its beginning to the current time, covering the reasons why it is still not widely adopted. The motivation and benefits about software reuse are enumerated in Section 2.2. Section 2.3 presents some impediments for software reuse adoption and Section 2.4 discusses the reusable software assets focusing on source code and means of its access. Finally, Section 2.5 summarizes the main concepts of the software reuse.

2.1. Introduction

Since late 1940s software development has witnessed a continuous stream of innovations from subroutines in the 1960s through to modules in the 1970s, objects in the 1980s, and components in the 1990 [Clements and Northrop, 2002]. Along the time, software projects became more complex and exigent, with constraints involving schedules, costs, and failures to meet the requirements defined by the customers or third parties [Ezran et al., 2002].

In the first NATO Software Engineering Conference in 1968, McIlroy [McIlroy, 1968], come up to the main concepts of the **software reuse** field,

however, at that moment, software reuse was faced as a mean for overcoming the current software crisis. McIlroy stated that a sub-industry of **reusable software components** was necessary to make the software industry well founded and able to tackle the running crisis in that decade. That was the first breathing of software reuse field to a software community, which motivated the rationale on how producing software from existing pieces of codes previously built.

Since McIlroy [McIlroy, 1968], a set of definitions about software reuse has been presented in the literature. For Frakes & Isoda [Frakes and Isoda, 1994] software reuse is defined as *the use of engineering knowledge or artifacts from existing systems to build new ones*. More detailed, [Ezran et al., 2002] state software reuse as *the systematic practice of developing software from a stock of building blocks, so that similarities in requirements and/or architecture between applications can be exploited to achieve substantial benefits in productivity, quality and business performance*. For instance, Krueger's general view [Krueger, 1992] will be adopted in this dissertation: *"Software reuse is the process of creating software systems from existing software rather than building them from scratch"*.

Indeed, software reuse is a very active field where a great amount of effort has been put in both industry and academy. Many case studies, research reports and surveys have been published but the truth is that software reuse is still considered in development given the absence of industrial large-scale success cases in the literature, as stated by Almeida et al. in [Almeida et al., 2005b]. Nowadays while some studies explore the possible reasons for the failure others prefer to use the results to evidence their mistakes and then propose improved mechanisms [Morisio et al., 2002] [Lucrédio et al., 2008]. The optimistic viewers outline the reuse benefits in terms of software development aspects: for them especially quality, productivity and cost reducing are key factors for achieving the level of success expected by the organizations [Lim, 1994] [Basili et al., 1996] [Frakes and Succi, 2001].

In the next Section the benefits and motivations for software reuse adoption in organizations are discussed.

2.2. The Motivation and Benefits of Software Reuse

As briefly discussed, the general benefits of software reuse are related to the increased productivity and the saved cost. **Productivity** gain is achieved when users are encouraged to reuse software assets instead of developing new ones, consequently, it results in less time being required to develop a system and hence an improved **time-to-market**.

In addition, the use of existing software assets may result in better **quality** since the defect density of tested software assets is lower than what can be expected in freshly created assets. This occurs naturally because error fixes accumulate from reuse to reuse which increases the **reliability** of a software system. Moreover, reusing existing software assets contribute to **effort reduction** since it demands fewer personnel. In most cases, smaller team sizes make projects easier to manage, leading to better communications and speed up software development. Additionally, the chances of **redundant work decreases** since developers get noticed about it previously built assets avoiding new development [Basili et al., 1996] [Sametinger, 1997] [J.S.Poulin, 1997] [Lim, 1998] [Sommerville, 2004].

Focusing on source code reuse, the literature has evidenced some reports which show that **40% to 60%** of **code** is reusable from one application to another, **60%** of **design** and **code** are reusable in business applications, **75%** of **program functions** are common to more than one program, and only **15%** of the **code** found in most systems is unique and new to a specific application [Ezran et al., 2002] [Almeida et al., 2005a]. According to Mili et al. [Mili et al., 1995] rates of actual and potential reuse range from **15%** to **85%**. With the maximization of tested, certified and organized reusable software assets, organizations can obtain improvements in cost, time and quality as will be explained in next sections [Basili et al., 1996]. Software reuse, thus, provides competitive advantage in terms of costs, time-to-market for a wide range of projects by utilizing the available resources in a more efficient way. However, this optimistic scenario is sometimes berried due to software reuse impediments discussed in the next section.

2.3. Software Reuse Impediments

Despite of its benefits, software reuse is not a common practice in most of organizations. In fact, systematic reuse of software has not universally delivered significant improvements in quality and productivity [Schmidt, 1999]. According to Sametinger [Sametinger, 1997], organizational, economical and technical impediments are some factors that avoid reuse in large scale by software organization.

Management impediments – The reuse infrastructure requires total management involvement since it generates cost and demand investment to the whole organization. Managers have to be in conformance with initial investments and be up-to-date of future savings acquired with reuse activity. In addition, managers must incentive the reuse practices through presentations, training programs as well as salary incentives to developers concerned in spend time in making components of a system reusable.

Organizational impediments – Systematic development, deployment and support of reusable software assets requires a deep understanding of the application developer needs and business requirements. Organizational structures must consider different needs for the “new” paradigm in software development, e.g., while a developer team consume assets previously built, other team may be arranged to product the ones. As the number of developers and projects employing reusable assets increases, it becomes hard to structure an organization to provide effective feedback loops between these constituencies.

Economic impediments – Achieving success in reuse activity leads important economic saving by software organization, however, it requires concrete investments in training, infrastructure, tools, process education [Poulin, 1997]. These initial costs does not guarantee early payback, therefore many organization avoid investments in a systematic reuse adoption. In addition, Poulin [Poulin, 1997] states that developing assets for reuse is more expensive than developing them for single use only because those requires higher levels of quality, reliability, portability, maintainability, generality and more extensive documentation are necessary.

Psychological impediments – Application developers may also perceive “top down” reuse efforts as an indication that management lacks confidence in their technical abilities. In addition, the NIH (Not Invented Here) syndrome is ubiquitous in many organizations, particularly among highly talented programmers. However, empirical studies [Frakes and Fox, 1995] have shown that developers are often willing to reuse if they know the right components to be reused [Isoda, 1995]. The real problem might be associated with the perception of the cost and the risks associated with reuse, including the location, comprehension, and adaptation of the components to be reused [Ye and Fischer, 2002].

In addition to **non-technical impediments**, reuse efforts also frequently fail because developers lack technical skills and organizations lack of competencies necessary to create and/or integrate reusable components systematically. The technical obstacles for software reuse include issues related to search and retrieval of components, legacy components and aspects involving adaptation [Schmidt, 1999].

Tooling impediments – In general, software organization do not have the necessary infrastructure tooling for cataloging, archiving, and retrieving reusable assets across multiple business units. Although it is common to search small classes or functions opportunistically from existing programs, developers often find it hard to locate suitable reusable assets outside of their immediate workgroups. This scenario invites the acquisition of information retrieval tools as well as reuse repository for managing the access to reusable assets [Mili et al., 1998].

Software Quality impediments – Not any asset is available for reuse, bad quality assets usually requires strong modification, documentation improvement and design enhancement as well. Assets focused on reuse are carefully specified, designed, implemented, tested and well documented. Instead of unprepared assets, they do not require complex modification when reused. This problem is commonly evidenced in use of legacy software because it entails preparation before be able to reuse, in addition, the efforts needed for understanding and extraction should be regarded.

Maintainability impediments – It is very difficult to find a component that works exactly in the same way that the developer wants. In this way, modifications are necessary and there should exist ways to determine their effects on the component and its previous verification results.

2.4. Reusable Software Assets

A reusable software asset is, broadly speaking, "*any cohesive collection of artifacts that solve a specific problem or set of problems encountered in the software development life cycle*". A reusable asset, which provides a solution to a problem for a given context, may have a variability point with a value provided or customized by the asset consumer, and rules for usage which are the instructions describing how the asset should be used [Lenz et al., 1987]. Artifacts are any work products from the software development lifecycle, such as requirements documents, models, source code files, deployment descriptors, test cases or scripts. In addition, software assets encapsulate business knowledge and are of high value to a company which generally requires legal terms of license [Ezran et al., 2002].

To produce reusable software assets, **Domain Engineering** play an important role for later use in the development of applications family. *Domain Analysis*, *Domain Design* and *Domain Implementation* are phases of Domain Engineering and have different responsibilities to build the application families [Almeida, 2007]. According to [Clements and Northrop, 1996], *Domain Analysis* is the process of identifying, collecting, organizing, and representing the relevant information in a domain, based upon the study of existing systems and their development histories, *Domain Design* takes the raised information and model family application considering its commonalities and variability so that they could be instanced for different applications and *Domain Implementation* embrace the whole process of identifying reusable asset components based on the domain model and generic architecture from domain knowledge gathered during domain analysis, and the generic architecture developed during the domain design.

In spite of the variety of the produced asset components, they are divided in two main types [Almeida, 2007]:

- **Vertical assets** which are specific to an application domain, for example, financial object models, algorithms and frameworks; and
- **Horizontal assets** which are straight to many different applications. They can be reused independently of the application domain answering to compatibility constraints. Examples of them include GUI objects, database access libraries, authentication service, and network communication libraries.

Besides these two kinds, assets may have different sizes and granularities such as a function or a procedure, a class, a group of classes, a framework and an application or a product. A reusable asset is potentially made up of many life-cycle products including: requirements and architecture definition, analysis model, design models, source code, test scenarios and test reports. Source code, in particular has a great importance because it holds the entire business logic of the software programs. In addition, source code has a high degree of reusability so that the functionalities carried out by its methods may be reused by other applications.

In the next Section, a special attention is given to the source code because it comprehends one of the key points of this dissertation.

2.4.1. Source Code Reuse

The importance of the source code is evidenced as for its historical appealing [McIlroy, 1968] as for being an indispensable artifact in any software development cycle. Since 1968 when the idea of software reuse, source code has gained particular attention in the reuse industry by the development of tools such as IDEs, dedicated search engines and reuse repositories to provide an environment in favor of code reuse.

In the software reuse industry, source codes tend to be encapsulated in components together with all necessary documentation in favor of its reuse. Ravichandran and Rothenberger [Ravichandran and Rothenberger, 2003]

present three software component reuse strategies: white-box reuse, black-box reuse, and black-box reuse with component markets:

- **White-box reuse** allows developers to modify the code to suit their needs. This maximizes reuse opportunities, but it is also a source of reuse problems due the fact that modifications in the code are not documented and versioned;
- **Black-box reuse** avoids these problems by not allowing the developers to modify the reusable components which they retrieve. This, however, dramatically reduces the reuse rate; and;
- **Black-box reuse with component markets** (i.e. obtaining components from a marketplace) can increase the reuse rate, as the developers can search from a larger set of components and thereby are more likely to find components fitting their requirements.

Ravichandran and Rothenberger [Ravichandran and Rothenberger, 2003] argue that the last strategy could be the “silver bullet solution” which makes software reuse a reality and advances software development to a robust industrial process. In spite of agreeing with them, the current scenario shows that software organization is not prepared for that. Currently, the lack of education on reuse obstructs the existence of teams exclusively turned to development of **black-box** components. Thus, until the companies achieve a maturity level on reuse, they have to concentrate its efforts with isolated reusable software assets such as source codes, requirement documents and design artifacts.

Some studies [Frakes and Fox, 1995] [Schmidt, 1999] [Glass, 2002] have discussed interesting questions about source code reuse such as: NIH (Not Invented Here) syndrome, programming languages paradigms and cost of modification. Frakes & Fox [Frakes and Fox, 1995] gave an expressive contribution by analyzing sixteen questions about software reuse using survey data collected from organizations in the U.S and Europe. They surveyed software engineers, managers, educators and others in the software development and research community about their attitudes, beliefs, and practices in reusing code and other lifecycle objects. The findings contributed to

breaking some of the myths related to software reuse. In the following, it is outlined a couple of question particularly related to source code:

- **NIH (Not Invented Here)** – The existence of the NIH syndrome was analyzed because there is a belief that programmers avoid working on third party codes. The research demonstrated that users prefer reuse rather than coding from the scratch, vanishing then the NIH syndrome;
- **Paradigms of Programming Languages** – The differences among programming languages were analyzed to verify if they affect reuse or not. Although OO languages prime for modularization, the study showed up the paradigm does not influence on reuse at all; and
- **Source Code Modification** - It was verified if modification of reused code is particularly error-prone, it is more efficient and effective to rewrite it from scratch. Glass [Glass, 2002] considers this idea has problem also, because of the complexity in maintaining existing software, the difficulty of comprehending the existing solution (even for the developer who originally built the solution) and the hard task in maintaining the documentation updated. Schmidt [Schmidt, 1999] credits this as a continuous problem considering the heterogeneity of hardware architectures, the diversity of OS and network platforms.

Despite still existing pros and cons positions about source code reuse, software reuse area depends of **systematic software development practice** [Ezran et al., 2002]. It is common to find solutions to aid the process of software development such as code generators, code scavenging, reengineering applications; however, this does not necessarily mean **systematic reuse** since it is often done in an *informal* and *opportunistic way*. Software developers usually know the exact location of codes with desired functionality when working in a project. According to Ezran and Morisio [Ezran et al., 2002], besides the management commitment and reuse education, the existence of a **technical reuse environment** is necessary in order to practically promote the reuse adoption throughout organization.

2.4.2. Accessing Reusable Software Assets

According to Pietro-Diaz [Prieto-Díaz, 1991] “*to access an asset, first you have to find-it*”, based on this, tools such as **reuse repositories** and **information retrieval tools** integrated to the developer environment are practical examples of effective tools for supporting the reuse activity.

Reuse repositories provide a wide range of assets useful for engineers to reuse during the software development. Most current reuse repositories are component-based; they provide a catalogue of software components categorized by some scheme. Reuse repositories are key tools for support reuse activity since developers may locate reusable components quickly and easily besides managing the whole life the component while present in the repository. Reuse repositories are expected to support some requirements such as searching, browsing, versioning, certification, status reporting, metrics extraction, notification and primarily asset maintained [Burégio, 2006].

Information retrieval systems provide effective means for accessing, sharing and retrieval knowledge embedded in assets in an efficient way. The facilities for accessing the asset information represent a potential mechanism to reuse amount of information available spread over legacy systems in the organizations or distributed in the Internet.

2.5. Chapter Summary

In this Chapter, the main concepts of software reuse were presented, discussing the origins, motivation and benefits, impediment factors for software reuse, types of reusable software assets with special attention to source code, in addition to mean for accessing them the in order to guide companies towards systematic software reuse.

Along the text it was observed that the development of a complete reuse infrastructure involves high initial costs as it includes well-established reuse processes, management engagement, education and tooling for supporting the reuse activity. The Chapter also outlined the technical and none-technical impediments for systematic reuse adoption. In addition, it was presented an

overview of reusable software assets, its variances with focus on source code. It was also discussed myths of source code reuse and the use of tools such as repository systems and information retrieval tools. In the next Chapter, a review on the information retrieval area with emphasis to mechanisms and technologies useful for promoting the reuse activity will be presented.



Information Retrieval

Information retrieval (IR) mechanisms can be regarded as an effective mean for promoting software reuse due to the amount of information available in the organizations or distributed on the Internet. In the context of software organizations, the available information is generally found in software assets used during the software life cycle such as requirement documents, source code and test cases. Information retrieval systems play a very important role by providing access to the available assets in a structured manner.

This Chapter presents a review on the information retrieval field outlining applications focused on the software reuse. It is organized as follows: Section 3.1 presents an introduction of the information retrieval field. Section 3.2 outlines a comparison between information retrieval and data retrieval. Section 3.3 discusses desirable aspects of search retrieval and preeminent challenges of the area. Section 3.4 introduces semantic treatments for facing information retrieval problems. Section 3.5 discusses technical support for information retrieval systems. Section 3.6 presents search retrieval issues focused on source code. Section 3.7 outlines the related works while Section 3.8 provokes a discussion about the works analyzed, and finally Section 3.9 summarizes the discussion on information retrieval and its application to the software reuse context.

3.1. Introduction

At the beginning, information retrieval mechanisms appeared to supply limited human capabilities in storage and retrieval [Baeza-Yates and Ribeiro-Neto, 1999].. In general, an user either does not have time or does not wish to spend time reading the entire document collection, apart from the fact that it may be

physically impossible for him to do so. Unfortunately manipulating information is not a trivial task; it requires support of technical mechanisms for structuring the data in order to be accessed in a reasonable performance. *Indexing* is an example of one of the most commons techniques for high speed in data accessing [Clarke and Cormack, 1995]. Nowadays, the indexing feature is largely applied over the information retrieval systems with complex algorithms; however, the attention is also turned to the quality of information retrieved. Relevant information credits the information retrieval systems and responds for their success and popularity in the trade market [Baeza-Yates and Ribeiro-Neto, 1999].

In spite of information retrieval concepts seems to be common to everyone, there are two distinct fields with particular characteristic that reveal the real identity of existing retrieval systems: *information retrieval* and *data retrieval* systems [Rijsbergen, 1979]. In the next Section, the dichotomy between both subfields will be explored in order to understand the differences and situate the proposed approach in one of those.

3.2. Information Retrieval versus Data Retrieval

The retrieval activity is distinguished between two models: *information retrieval (IR)* and *data retrieval (DR)*. In spite of the boundary between both is small for some authors, there are clear concepts that differentiate existing information retrieval systems.

According to Rijsbergen [Rijsbergen, 1979], *data retrieval* is normally looking for an exact match while in *information retrieval* is more interested in finding those items which partially match the query and then select from those a few of the best matching ones. Consequently, *data retrieval* is more sensitive to errors considering that a single erroneous returned item among a thousand retrieved means total failure. In opposite, an *information retrieval* search might be inaccurate and eventual errors do not comprise the search capability at all. This happens because *information retrieval* usually deals with unstructured natural language texts that can be semantically ambiguous while *data retrieval*

system deals with defined data structure such as relational database. Table 3.1 summarizes differences between both modes of retrieval.

Table 3.1. Information Retrieval x Data Retrieval

Characteristic/ Methods	IR	DR
Exact Match		x
High Error Sensitiveness		x
Semantic Treatment	x	
Search in unstructured data	x	
Deductive Inference		x
Inductive Inference	x	
Controlled Syntax Query		x
Natural Language Query	x	
Commonly Academic Development	x	
Outstanding Commercial Products		x

In fact, an information retrieval user is concerned more with retrieving information about a subject than with a precise data to satisfy his query. To be effective, IR systems must somehow “*interpret*” the contents of the information items (documents) in a collection and rank them according to a degree of relevance to the user query. This “*interpretation*” of the item content involves extracting *syntactic* and *semantic* information in order to match the user information need. Baeza-Yates and Ribeiro-Neto [Baeza-Yates and Ribeiro-Neto, 1999] outlines the difficulty is not only knowing how to extract this information but also knowing how to use it to decide relevance. Thus, the notion of relevance is at the center of information retrieval. In fact, the primary goal of an IR system is:

“...retrieve all the relevant documents at the same time retrieving as few of the non-relevant as possible...” [Rijsbergen, 1979].

The inference used in *data retrieval* system is the simple deductive kind while in *information retrieval* it is more common to use inductive inference; here, to define relations it depends on the degree of certainty. This peculiarity

leads one to describe *data retrieval* as deterministic whereas *information retrieval* as probabilistic. For that reason, is common to find *information retrieval* systems that make use of probabilistic models [Sebastiani, 2002] to carry out inferences to satisfy the entire search process.

One more distinction can be made in terms of query language, where in *data retrieval* is normally done of artificial kind, with controlled and restricted syntax and sometimes proprietary, e.g. the particular query syntaxes for relational databases. On the other hand, in *information retrieval* favors to use natural language although there are some exceptions, e.g. the keyword query used in traditional web search engines. In addition, the query used for *data retrieval* systems is more detailed when expressing what is wanted, in opposite, *information retrieval* it is invariably incomplete. This fact justifies the high sensitiveness to errors in *data retrieval* systems.

An “*empirical*” distinction is attributed to the great majority of automatic *information retrieval* systems have great support of academy experiments frequently developed in university laboratories, however, it does not mean that not exist commercial solutions as well. *Data retrieval* systems, in general, have a commercial propose with intuit of sell its advantages such as performance and innovative functionalities.

The proposed solution in this dissertation will integrate an *information retrieval system* due the fact it does not require any syntax control in query formulation; the search is performed on a collection of unstructured documents; its result relevance depends on semantic assistance; it makes use of probabilistic model for augment item comprehension, and originally it comes from academic initiative.

In the next Section it is emphasized how relevant information credits the information retrieval systems and drives them to the success through the IR community and trade market as well.

3.3. The Relevant Information

In a “*perfect retrieval*”, an information system search though a set of documents reading all them, retaining the relevant ones and discarding all the others. This envisioned solution reveals a truly and continuous “*search for*

treasures”; the great challenge is to distinguish the relevant documents from the non-relevant ones. According to [Baeza-Yates and Ribeiro-Neto, 1999], this will be the threshold to determine the success of an information retrieval tool which intends to reach an outstanding position in the market.

To illustrate the importance of relevant returns, consider a company that has invested a lot of money in a very fast search engine which answers a single query in some milliseconds; now consider that after a search, the first 10 results do not satisfy the current query and the information desired is encountered only in the eleventh item. Analyzing the whole process, the user was obligated to look for the information throughout the first 10 ten items until reach the exact piece of information. The high performance achieved for answering the input query became irrelevant *when comparing with the time spent for reaching the information desired*. Moreover, it is not difficult to evidence that occurred a duplication of effort considering the user had to seek information again after the items browsed.

In the best case, an information retrieval system is supposed to extract the information from the text (both syntactic and semantic) and use it to decide whether each document is relevant or not to a particular request. While no reference solution is showed up, much research has been carried out in the academy as well as in the organization. Intelligence Artificial disciplines have provided techniques such as automatic text categorization [Lam et al., 1999], semantic inference and ontology reasoning [Bruijn, 2003] in order to apply them in information retrieval systems aiming to augment the search relevance. In a nutshell, these techniques try to overcome information retrieval problems such as query formulation and code comprehension. Next Section depicts these problems and presents some ongoing effort to solve them.

3.4. The Semantic Challenges of Information Retrieval Systems

By analyzing the entire life cycle of search retrieval, two significant troubles are quite clear: *use of inappropriate words in query formulation* and *arbitrary retrieval*. For Rijsbergen [Rijsbergen, 1979], advances in these two main fields may conduct information retrieval systems to achieve an search proficiency.

Undoubtedly, one of the most difficult parts of search strategy is deciding upon the keywords to use. A common mistake is not providing enough information that represents the realistic user request. Inappropriate keywords increase the probability of unexpected results that mismatch the user intention [Baeza-Yates and Ribeiro-Neto, 1999]. According to [Ye and Fischer, 2002], such distance between *user need* and the *computational understanding* of the query is called “*Semantic Conceptual Gap*”. More precisely, the *Semantic Conceptual Gap* occurs when ambiguous formulation of a contextual knowledge in a powerful language (e.g. natural language) is not properly translated in a computational representation in a formal language (e.g. programming language).

Making an analogy to web search engines, ambiguous query formulation such as “*window*” may return links to “*house window store*” as well as “*Microsoft Windows OS*”. In fact, the **semantics** of a query depends on the context it is regarded within. Therefore, it is rational to outline formal representation of real world tasks require translation of the *contextual knowledge* into *understandable computer operations*.

In addition to problems in query formulation, *arbitrary retrieval* is also a significant issue under continuous worry [Lam et al., 1999]. It concentrates its efforts in a creation of formal methods to enhancing retrieval effectiveness such as the frequency of occurrence and co-occurrence of index terms in the relevant and non-relevant documents.

In fact, users always wish the best returns independently of their query; from that point, technologies have to evolve in order to supply such exigencies [Harter, 1992]. From this premise, innovative search techniques have been developed to provide additional search options in order to search effectiveness. Next Section covers some functionalities shared by many applications provided of some semantic assistance.

3.5. Technical Support for Information Retrieval Systems

Many initiatives to overcome the *semantic conceptual gap* and *arbitrary retrieval* have been proposed. In this Section is showed a range of techniques

utilized for query formulation as well as text comprehension methods for augmenting semantics in the retrieval process.

3.5.1 Techniques for Query Formulation Support

Specifically talking about the *query formulation*, the proposed techniques aim to enhance the original query by adding relevant information which might be useful or relevant to the user.

In [Devanbu et al., 1991], an information retrieval system that utilizes **hierarchical categories** to identify components was presented; Prieto-Díaz [Prieto-Díaz, 1991] in its proposal have applied the concept of multiple **facets** which encapsulate aspects, properties and characteristics of a software component. Belkin and Croft [Belkin and Croft, 1992] have applied **information filtering** to remove redundant or unwanted information from the returned collection. The **query by reformulation** mechanism was utilized by Henninger [Henninger, 1993] in which the original query was expanded by user to match additional software component. Components constraints, such as **signature matching** and **formal specification**, were used by some reuse repository systems for component retrieval [Podgurski and Pierce, 1993] [Zaremski and Wing, 1995].

In 2002, Ye and Fischer [Ye and Fischer, 2002] have utilized **context-based** retrieval where the returned components took into account class syntax structures while user coding. Sugumaran and Storey [Sugumaran and Storey, 2003] have used an **domain ontology** in a source code search engine to correlate the domain-specific concepts with indexed classes in the repository. Calado and Ribeiro-Neto [Calado and Ribeiro-Neto, 2003] have proposed an approach for the formulation of **approximate queries** and ranked the results according to the user's information need. This approach includes a new algorithm for the calculation of semantic similarities among concepts that represent attribute values in a database based on a vector space probabilistic model. Oyama et al. [Oyama et al., 2004] have proposed a **domain-specific** web search engine that is based on the idea of keyword spices: Boolean expressions that are added to the user's input query to improve the search performance. In 2007, Vanderlei et al. have applied the **folksonomy**

[Vanderlei et al., 2007] mechanism in a source code search engine in order to assist users to search codes through a tag cloud. Hotho et al. [Hotho et al., 2006] have proposed a formal model and a new search algorithm for **folksonomy**, called *FolkRank*, that exploits the structure of the folksonomy to find out communities and ranking the search results.

As seen many techniques have been proposed to overcome the *semantic conceptual gap* and *arbitrary retrieval*. In general, these initiatives are useful when combined with *classification* schemes in order to augment the chances of successful retrieval.

3.5.2 Classification Schemes for Component Retrieval Support

The query capabilities can be combined with *classification* schemes in order to augment the chances of successful retrieval [Ugurel et al., 2002]. Classification (or categorization) schemes may be applied to automatically extract information about the asset content and use this to increase its decision in the retrieval process. Although manual categorization is possible, it is very expensive to maintain and discouraged for dynamic systems, therefore automatic classification techniques are useful instruments against arbitrary retrieval.

Automated classification of texts into topical categories has being researched since the early '60s, however, in '90s, with advent of the internet which promoted the booming in production and availability of on-line documents, this technique has witnessed an increased and renewed interest. From that time, automatic categorization began to be seen as the *meeting* point of machine learning and information retrieval [Sebastiani, 2002] .

In general, typical text classifiers learn from a knowledge base which automatically builds a classifier (also called the rule, or the hypothesis), this analyzes document characteristics and classify them in one or more predefined categories. Information retrieval systems usually apply automatic text classification to raise additional information about the text content. This meta-information is useful to compose the index structure providing a higher degree of semantics. The advantages of this approach are the considerable savings in terms of expert manpower and domain independence. In the following, the

main approaches of automatic text categorization within the general machine learning paradigm are described:

- **SVM - Support Vector Machines (SVM)** is based on a vector space where the purpose is to find a decision surface that “best” separates the data points in two classes which represent the categories. The objective is to finding out the maximum separation (margin) between the two classes, i.e., to pick the hyper plane so that the distance from the hyper plane to the nearest data point is maximized [Lam et al., 1999];
- **KNN - The *k*-nearest neighbor** classification is based on the nearest neighbor algorithm. Given a test document, the system finds the *k* nearest neighbors among the training documents and uses the categories of the *k* neighbors to weight the category candidates. By sorting the scores of the candidate categories, a ranked list is obtained for the test document [Lam et al., 1999];
- **Neural Network (NNet)** - Neural network techniques have been applied successfully to speech recognition, image analysis and cluster classification among others [Lam et al., 1999]. For classification, *NNet* utilizes separate neural network per category, learning a non-linear mapping from input words (or more complex features such as singular vectors of a document space) to a category; and
- **NB - Naive Bayes (NB)** - Probabilistic classifiers are commonly studied in machine learning [Lam et al., 1999]. The basic idea in NB approaches is to use the joint probabilities of words and categories to estimate the probabilities of categories given a document. The naive part of NB methods is the assumption of word independence, i.e., the conditional probability of a word given a category is assumed to be independent from the conditional probabilities of other words given that category. This assumption makes the computation of the NB classifiers far more efficient than the exponential complexity of non-naive Bayes approaches because it does not use word combinations as predictors.

A deep analyze of each method describing the robustness and performance details including benchmarks results can be better explored in [Yang and Liu, 1999] and [Sebastiani, 2002].

The literature has shown some success cases that prime for code comprehension and classification. Ugurel et al. [Ugurel et al., 2002] have employed vector machines for source code classification in a two-phase process, consisting of programming language classification followed by topic classification. Lam et al. [Lam et al., 1999] have proposed an categorization approach derived from a combination of machine learning technique and a text retrieval technique known as *retrieval feedback* where the queries performed are refined without user interference.

In agreement with the theory presented, this semantic proposal applies text categorization for increasing source code comprehension. The goal is to use machine learning technique in order to identify domains of technology handled by source codes.

3.6. Source Code Retrieval for Reuse Activity

According to Henninger and Belkin [Henninger and Belkin, 1996] little attention has been given to software components retrieval. Most research on information retrieval systems has centered its effort in methods for effectively retrieve relevant documents. However, specifically talking about source code, there are arbitrary rules of grammar that are different from natural language as well as semantic issues attributed to its functionalities. Therefore, information retrieval systems away to those particularities will find difficult to search source code precisely.

Unlike documents written in natural languages, source code is unambiguous to the compiler and has exact syntactic structures [Ugurel et al., 2002], however, it is not common to find code searchers equipped with a compiler to help in query processing. Furthermore, the syntax itself does not respond for semantic of a particular piece of code. The semantic outlined refers to the functionalities covered by the class through its methods. Therefore, the lack of “*semantic analyze*” represents a significant barrier to locate source codes. To illustrate this problem, Ye and Fischer [Ye and Fischer, 2002] plausibly observed that if a software developer wants to *draw a circle*, it is necessary to know that the method “*drawOval*” in the Java class library perform

the functionality desired or at least recognize this method belongs to the *java.awt package*. According to [Henninger and Belkin, 1996], a conventional wisdom is that software components must be organized in some manner so they can be found. One possible direction for augmenting source code representation is to construct a classification scheme where the retrieval process can be accomplished by choosing the right category. On the other hand, the negative point of this approach is that finding the right category may be a difficult task because it involves a detailed understanding of the classification scheme.

Be aware to these problems intrinsic to source code, it configures the first advance towards effective source code search engines. Based on this, Garcia et al. [Garcia et al., 2006a] have elaborated an review about the state-of-the-art of code search engines. In this study, Garcia et al. outline essential aspects that must be considered by any code searches towards an effective support of the reuse activity. In addition Garcia et al. have presented commercial and non-commercial search engines and how their techniques have evolving by the time.

Based on Garcia et al. [Garcia et al., 2006a] study and other contributions, next Section presents a range of information retrieval tools which make use of different mechanisms in order to perform searches more efficiently and consequently improve the reuse activity.

3.7. Information Retrieval Systems for Component Reuse

As previously mentioned in Chapter 1, an essential step in software reuse is to find assets previously built. *Source Code Search Engine* has long been contributing to reuse activity since early 90's when the first works appeared [Ezran et al., 2002]. Currently, such engines have gained attention and attracted investments from giant software companies. This Section reviews outstanding code searchers supported by different mechanisms in order to enhance its search efficiency.

In 1991, Prieto-Díaz proposed the utilization of a **facet-based scheme** to classify and consequently retrieve software components. The goal was to manually describe components according to their different characteristics, unlike the traditional hierarchical classifications, where a single node from a

tree-based scheme hide the component particularities [Prieto-Díaz, 1991]. However, researchers such as [Maarek et al., 1991] argue that classifying components manually is susceptible subjective, so that two different people may choose different keywords or facets to describe the same asset. In this sense, Maarek et al. tackled the similarity problem by **automatically clustering** artifacts from free-text descriptors, terms or phrases that best describes a component. In 1994, Henninger [Henninger, 1994] presented the *CodeFinder*, a code searcher that uses **query-construction** methods for assisting users to define their needs when they do not know the exact terminology.

In late 90's, the Software Engineering Institute at Carnegie Mellon University developed an information retrieval system called by *Agora*. Search and retrieval in *Agora* begins when a searcher enters **keyword query** and optionally specifies the **type of component**. These terms and other criteria are searched against the index collected by the search agents. The result set for the query is sent back to the user for inspection. Each result includes meta-information including the URL of the component and the searcher can then refine or broaden the search criteria based on the number and quality of the matches [Seacord et al., 1998].

In 2000, [Thomason et al., 2000] proposed the *CLARiFi*, a component-based system that provides an classification schema that identifies **component properties** important in the selection for a given task. In 2002, Ye and Fischer [Ye and Fischer, 2002] presented the *CodeBroker*, a **context-based** code searcher to retrieval source code in accordance with the developer environment. They proposed a process called information delivery (or active search), which consists in anticipating the software engineer's needs for components. The process is performed by monitoring the activities of the software engineer, e.g. codification or documentation, and automatically searching for the components. In 2003, Sugumaran and Storey [Sugumaran and Storey, 2003] presented A *Semantic-Based Approach to Component Retrieval* to meet user's requirement taking account domain models containing the objectives, processes, actions, actors, and, an **ontology** of domain terms, their definitions, and relationships with other **domain-specific terms**. Holmes and Murphy [Holmes and Murphy, 2005] proposed the *Strathcona*, an Eclipse plug-in that finds source code examples through a search based in six different **heuristics**. The retrieval

takes account the **code structure similarity** between the developer codes under writing against those indexed in the repository.

In 2005, Garcia et. al [Garcia et al., 2006b] presented a **keyword** and **facet-based** component search engine called *Maracatu*. The client-server architecture allowed a client Eclipse plug-in searches Java source code from CVS repositories indexed in the server side. In 2006, Mascena et al. [Mascena, 2006] proposed a first evolution of Maracatu: an integrated reuse environment called *ADMIRE*, that was based on the same concept of **information delivery** proposed by Ye and Fisher [Ye and Fischer, 2002]. On its work a new **reuse metric** also was proposed with the goal of monitoring the reuse activities and allows the software engineer to make corrective actions across the development process. The second evolution of Maracatu was developed by Vanderlei et al. [Vanderlei et al., 2007]. In this version, Vanderlei et al. presented the use of **folksonomy**, a **tag-based** mechanism where developers manually tag source codes with related terms.

After 2005, keyword-based web search engines focused on source code retrieval also started to appear on the Internet. *Koders* web search engine [Koders, 2006] automatically connects with different version control systems (e.g., CVS and Subversion) to search source code, being able to recognize approximately 30 programming languages and 20 software licenses. Likewise *Koders*, *Krugle* [Krugle, 2006] helps professional developers solve their programming problems by searching for many different types of programming languages. With the same purpose, *Merobase*² was proposed to allow users to find, remember and share components on the Internet. In contrast with first-generation code search engines, *Merobase* treats source code modules as first class abstractions rather than chunks of text. In particular, *Merobase* specializes in finding components based on their interface (or API) rather than the strings in their source code. Not different *Google Code* [Google, 2006] search public source codes for function definitions and sample code in many types of programming language types as well. In continuation, Google has recently released the *Google Code Search data API* that allows developer to create client and web applications to search public source code on the Internet. In addition, *Google Code* allows users perform search through specific facets.

² <http://www.merobase.com>

In 2006, the RiSE group [Almeida et al., 2004] and C.E.S.A.R³ (Recife Center for Advanced Studies and Systems) have sponsored the development of a commercial version of *Maracatu* search engine [Garcia et al., 2006b]: *B.A.R.T*, the *Basic Asset Retrieval Tool*. It is a robust information retrieval system with three user interfaces [Almeida et al., 2004]: Eclipse and Ms/Word plug-ins and a brand new Web Interface for searching software assets. The B.A.R.T agenda envisages the addition of other search techniques - beyond the current keyword-based - such as facet-based, folksonomy, context besides innovative technologies related to data-mining and artificial intelligence. The semantic layer proposed in this dissertation represents the continuation of RiSE research agenda in favor of relevant search and retrieval. The complete description about how introducing semantics in the B.A.R.T search engine is entirely detailed in Chapter 5.

The search engines presented so far have employed their efforts in source code retrieval without the use of semantic web benefits. Nevertheless, the next Chapter outlines how semantic search engines have exploited ontologies to increase the recall and precision of its results.

3.8. Information Retrieval Discussion

Although the effort for best retrieval introduced by Prieto-Díaz [Prieto-Díaz, 1991], the facets may vary according to the project context being useful for ones but not for others. The main obstacle here is to choose the most adequate facets to compose the scheme in accordance with the project nature. Maarak et al. [Maarek et al., 1991] and the CLARiFi [Thomason et al., 2000], a try to solve the Prieto's dilemma by automatically clustering artifacts from free-text descriptors, terms or phrases that best describes a component. These mechanisms should be complemented by the CodeFinder [Henninger, 1994] mechanism that uses query-construction methods for assisting users to define their needs when they do not know the exact terminology. The combination of the last techniques probably would result in a powerful search engine with increases in the precision.

³ <http://www.cesar.org.br>.

To complement the presented approaches, Sugumaran and Storey [Sugumaran and Storey, 2003] introduces the use of ontologies that could be combined with Henninger mechanism in order to user proper terms in the query formulation. Proposals such as Maracatu [Garcia et al., 2006b] and Strathcona [Holmes and Murphy, 2005] must have special attention by the fact that they take advantage because were developed to be in the user environment with facilitates their usage. The use of folksonomy technique as implemented by Vanderlei et al. [Vanderlei et al., 2007] may be regarded as important step towards free hand classification by use of tags. This feature has been appearing with more frequency in many important information retrieval tools around the world.

Web search engines such as Krugle, Merobase and Google never must be ignored due the fact that they are available to everyone, however, all of the techniques presented so far must be presented in order increase their attractiveness in terms of relevance.

3.9. Chapter Summary

This Chapter introduced the information retrieval field and discussed its relationship to software reuse. It was described the differences between data retrieval and information retrieval system, some question about relevance search, techniques for assisting searching and recovering and related works in favor of software reuse. Next Chapter outlines how semantic web technologies and ontology may be useful for supporting information retrieval systems with semantic assistance.



The Semantic Web

With the advent of the Internet, vast amounts of knowledge was spread through the web pages for human consume, however, the Semantic Web field has explored means of how to make that information also machine-understandable [Berners-Lee et al., 2001]. Planned to be an extension of current World Wide Web, the Semantic Web aims to express the web content in a format that can be read and used by software agents, thus permitting them to find, share and reuse information easier. This Chapter makes a review of Semantic Web concepts, technologies and outstanding projects that have been taking advantage of this technology.

The Chapter is organized as follows: Section 4.1 presents an introduction about the advent of semantic web as well as its purpose. Section 4.2 introduces the ontological concepts which are important for the basis of the Semantic Web field. Section 4.3 outlines the real-world ontology aspects such as building time and costs. Section 4.4 depicts the ontology engineering process. Section 4.5 focus on ontology levels from different types of ontologies. Section 4.6 reports a review about the most outstanding semantic markup web languages. Section 4.7 presents a set of successful semantic projects which employ ontologies to other necessities. Section 4.8 outlines semantic search engines and discusses their benefits to the information retrieval area and finally Section 4.9 finishes the Chapter with a brief summary about what was presented.

4.1. Introduction

The World Wide Web has dramatically increased the availability of electronically available information. From 2003, more than 500 million computer users around the world access around 3 billion online documents

daily, and these numbers are expected to grow exponentially as organizations become more geographically dispersed and organized around virtual teams [Bruijn, 2003]. The web brings a huge volume of information available in documents, images, and video - all forms that require human intelligence to understand and process. To a computer, this information is just data, which can be stored, displayed, compressed, and transmitted to other computers. No understanding about the content is assured and therefore no logical relationship may be inferred by computer. The **Semantic Web** activity at the *World Wide Web Consortium* ⁴ aims to augment the current web with information that a computer may process and understanding its meaning.

Initially, the web was designed as an information space, with the goal that it should be useful not only for human-human communication, but also that machine would be able to participate and help. Nevertheless, the current web is under an “unintelligent” structure where the pages are concerned only in how to present the information and not for its self communication. According to Lee [Berners-Lee et al., 2001], to achieve this stage, the web has to fulfill some requirements, such as data structuring and semantic configuration in order to become understandable by the computers. Applying domain ontologies to web documents will allow data be **shared** and **reused** across application, enterprise, and community boundaries. This semantic technology may then become web documents “*machine-understandable*”.

4.2. Ontology

According to Webster’s Revised Unabridged Dictionary ⁵ the word “**ontology**” means: “*The department of the science of metaphysics which investigates and explains the nature and essential properties and relations of all beings, as such, or the principles and causes of being.*” In spite of such definition comes from the field of philosophy, the term was adopted also by Artificial Intelligence community. In computer science, a famous definition was introduced by Gruber that said:

⁴ <http://www.w3.org>

⁵ <http://machaut.uchicago.edu/websters>

"Ontology is a formal and explicit specification of a shared conceptualization"
[Gruber, 2002]

This short statement reveals the great role of ontology: formalize and establishing concepts and its relationships. To be “*formal*” means the ontology should be machine understandable; “explicit” because all concepts and constraints used are explicitly defined; “specification” because it represents the conceptualization in a concrete form; “shared” indicates that the ontology captures consensual knowledge and “conceptualization” means an abstract simplified view of the world that is desired to represent for some purpose [Gruber, 2002].

4.3. Real-world Ontology Aspects

In the late 1990s, with the advent of the Internet, the use of semantic markup ontology languages became widespread in areas such as data integration, knowledge management and information retrieval [Bruijn, 2003]. Behind its use some social and technical issues are considered in the following:

- **Human and machine link** - According to Fensel [Fensel, 2001], the main advantage about ontologies usage is the interlink between human and machine understanding through formal, real-world semantics and consensual terminologies, interweave human and machine understanding. Edgington [Edgington et al., 2004] credits its usage due to the capacity of knowledge sharing and reuse among both human and computer agents;
- **Empirical facts** - A typical reason for constructing ontology is to provide a common language for sharing and reusing knowledge about phenomena in the world of interest. It is important clearly establish the following distinction: on one hand, there is the ontology itself, which specifies concepts used in a domain of endeavor, concepts whose existence and relationships are true by definition or convention. On the other hand, there are empirical facts about these concepts and relationships. They are not part of the ontology, although creating a

general ontology characterizing the conduct of knowledge management [Holsapple and Joshi, 2002];

- **Cost** – To build a domain ontology definitely is not a cheap activity, Simperl et al. [Simperl et al., 2006] defend that the cost for applying ontologies in commercial applications depends on the availability of appropriate methodologies guiding the ontology development process and the time allocated to requirement engineers for contributing on the ontology validation. In addition, the money spent for training generally is high once ontologies are recent and still far from the software engineers knowledge. Moreover, the energy required to create, document and maintain ontology sometimes can be an inhibitor factor depending on the size and the complexity; and
- **Building Time** - An ontology seen as an artifact is under responsibility of the *requisition engineering*, because during the phase the knowledge of the context is well comprehended, thus the biggest effort for defining ontology is applied during the first phase of software development cycle. Developing a knowledge base has to respect some constraints, follows a methodology, pass over some steps and refine it. Therefore, the ontology creation is not supposed to be an ad-hoc activity, thus, a methodology is necessary to be followed and tools for supporting its process.

4.4. Ontology Engineering

Ontological engineering has gained increasing attention over the last few years, as researchers have recognized ontologies are not just for knowledge-based systems that need model desired entities of the world [Devedzić, 2002].

Ontological engineering encompasses a set of activities carried out during **conceptualization**, **design**, **implementation** and **deployment** of ontologies [Devedzić, 2002]. It provides the effective support of ontology development and use during its life cycle-design, evaluation, validation, maintenance, deployment, mapping, integration, sharing and reuse [Gomez-Perez et al., 2004]. Nevertheless, building ontologies is difficult, time-

consuming, and expensive, particularly if the ontology design is formal enough to support automated inference.

According to [Gomez-Perez et al., 2004], some methodological steps have to be followed to create ontology:

- **Purpose identification and requirements specification** - in this phase, the competence of the ontology is defined. To confirm that, it is necessary the agreement by multiple parties (persons and software systems) to adopt a particular domain of interest, even though they do not necessarily have the same experiences, theories, or prescription about that domain;
- **Capture Ontology** - the goal is to capture the conceptualization of the domain based on the ontology competence. The relevant concepts and relations should be identified and organized. In this phase, the taxonomy starts to be raised;
- **Formalize Ontology** - this phase is responsible for explicitly representing the conceptualization in a formal language. One of the most used tools for supporting this task is Protégé⁶. [Noy et al., 2001] present a graphical tool for ontology editing and knowledge acquisition that can be adapted to enable conceptual modeling with new and evolving Semantic Web languages like RDF and OWL;
- **Integrate existing ontologies** - during this phase occurs the integration among the current ontology with existing ones, in order to reuse established conceptualizations. Even though this activity is conceptually useful, it may be skipped in cases when the ontology subject is quite restrict to a very specific domain;
- **Evaluate Ontology** - at this point, the ontology is checked whether it satisfies the specification requirements. The knowledge management requires a continuous system of interaction and iteration with the knowledge owners to validate existing knowledge [Edgington et al., 2004]; and

⁶ <http://protege.stanford.edu>

- **Document Ontology** - during this phase, the entire ontology development must be documented, including purposes, requirements and motivating scenarios, textual descriptions of the conceptualization, the formal ontology and the adopted design criteria.

All of the steps are strongly advisable to be accomplished in order to formally build an ontology. These are the expected entailments to be fulfilled in organization with desire to publish and share their ontologies; on the other hand, this formal process may be tailored to adhere to organizational process of software development. Therefore, it is fairly compressively that steps could be added and more artifacts produced for documentation [Devedzić, 2002].

4.5. Ontology Levels

Ontology can embrace a range of knowledge since the most specific ones until the more common shared by diverse communities. The stricter ontologies usually describe specific concepts about a domain, in opposite, the top-level ones describe general concepts that are common across different domains.

The generality means the extensiveness of the ontology, for example, while some ontologies try to capture all terms in natural language, others are very specific to certain domains. The expressiveness of ontology is measure by the degree of explication of the (meta-) knowledge, which is captured. The more constraints are, the more expressive it is, since it captures the knowledge of the domain on a more detailed level [Devedzić, 2001]. In the following, the ontology levels will be presented according to Guarino [Guarino, 1998] which identifies three layers of knowledge, corresponding to three different types of ontologies, based on their levels of generality, namely:

- **Upper Ontologies or Top-level Ontologies** - describe general concepts, independent of any particular domain or task. Upper-level ontologies capture mostly concepts that are basic for the human understanding of the world such as “Thing” and “Behavior”. They are

“grounded” in (supported by, wired to) the common sense that makes it difficult to formalize a strict definition for them;

- **Domain ontologies and task ontologies** - describe, respectively, generic concepts for a particular domain and generic concepts for a generic task. For example, in the natural sciences (Mathematics, Physics, Chemistry, Biology, Medicine) the knowledge is easy to formalize because it is more or less systematic — it could be expressed using well-defined scientific terms [Bruijn, 2003]; and
- **Application ontologies** - describe concepts depending on both the domain and the task [Bruijn, 2003].

The generality of the level is fundamental to determine its degree of reuse: the higher level an ontology has, the higher its reuse is.

4.6. The Mark-up Ontology Web Languages

The current *World Wide Web* (WWW) contains large amount information which is expanding at a rapid rate. Most of that information is represented under *Hypertext Markup Language* (HTML), which is designed to allow web developers display information in a format suitable for human viewing through web browsers. On the other hand, the HTML format does not provide enough infrastructure that enable software programs to “understand” the information content. Then, the World Wide Web Consortium (W3C) has developed the *Extensible Markup Language* (XML)⁷: a metadata-based format which allows information be more accurately described. A XML document is more meaningful than the HTML one with respect to the information objects represented by text. The markup itself is a form of “metadata”, explaining what the constituent elements are (by name), and how these information objects are structured into larger coherent units. This markup structure allows computers to navigate and query over the information content [Berners-Lee, 1996] [Farrugia, 2003]

⁷ <http://www.w3.org/XML>

In spite of the advances in relation to HTML, the XML has a limited capability to describe the relationships and probabilities with respect to semantic of its objects. These characteristics, however, are naturally managed by domain ontologies; based on this, the W3C community extended the XML to address ontologies for displaying the semantics of the objects that are being defined. This initiative has promoted the advent of *Mark-up Ontology Web Languages*: a formal data model which supports the semantics of the entities as well as allows applications to process and manipulate its content [Berners-Lee, 1996] [Farrugia, 2003].

Next Section presents the most outstanding *Mark-up Ontology Web Languages*.

4.6.1. Resource Description Framework (RDF)

The *Resource Description Framework* or *RDF* is a general-purpose language for representing information in the Web [McBride et al., 2004]. The RDF data model distinguishes between resources, which are object identifiers represented by URIs, and literals, which are just strings. It is based upon the idea of making statements about resources in the form of *subject-predicate-object* expressions, called *triples* in RDF terminology. The subject denotes the resource, and the predicate denotes traits or aspects of the resource and expresses a relationship between the subject and the object. For example, one way to represent the notion "*The house has the color white*" in RDF is as a triple of specially formatted strings: a subject denoting "the house", a predicate denoting "has the color", and an object denoting "white". In *RDF diagrams* (Figure 2.1), resources are always drawn as ovals, and literals are drawn as boxes.

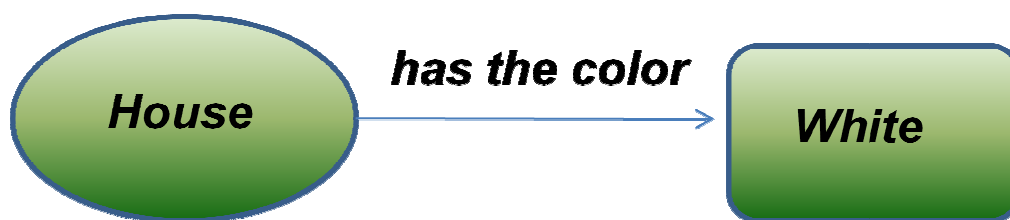


Figure 4.1. RDF diagram

In a sense, RDF is a practical and viable resource for establishing interoperation between Web applications. Being object-oriented, it has a more suitable data model for exchanging information than XML, and it is extremely flexible for defining new vocabularies. According to Decker, RDF is a promissory technology for the next phase in the development of the Web, when vocabularies and vocabulary marketplaces will become more important [Decker et al., 2000].

4.6.2. RDF Schema

RDF Schema⁸ is a semantic extension of RDF which provides mechanisms for describing groups of related resources and the relationships between them. These resources are used to determine characteristics of other resources, such as the domains and ranges of properties. Classes in RDF Schema are like conceptual classes in object oriented programming languages. This allows resources to be defined as instances of classes, and subclasses of classes [McBride et al., 2004].

4.6.3. Ontology Inference Layer (OIL)

The *Ontology Inference Layer* ⁹ is a proposed representation of machine-accessible semantics [Connolly et al., 2001]. Designed to be compatible with existing *World Wide Web Consortium* (W3C) standards, including XML and the RDF, OIL exploits the modeling primitives of RDF Schema. OIL is a proposal for a web-based representation and inference layer for ontologies, which combines the widely used modeling primitives from frame-based languages with the formal semantics and reasoning services provided by description logics. OIL still offers some levels of complexity designed to different sorts of descriptions; each level adds new functionalities and complexity to the one

⁸ <http://protege.stanford.edu/overview/protege-owl.html>

⁹ <http://www.ontoknowledge.org/oil>

below it. They are: *Core OIL*, *Standard OIL*, *Instance OIL* and *Heavy OIL* [Fensel, 2002]. Some open-source projects contribute to OIL dissemination such as *OilEd* that is a ontology editor which allows the user to build ontologies using OIL. The intention behind OilEd is to provide a simple, freeware editor that demonstrates the use of, and stimulates interest in, OIL. In the same way, *OntoEdit* ¹⁰ is an *Ontology Engineering Environment* supporting the development and maintenance of ontologies using graphical support [Sure et al., 2003].

4.6.4. Ontology Web Language (OWL)

The *Web Ontology Language* (OWL) ¹¹ is the most recent development in standard ontology languages, endorsed by the World Wide Web Consortium (W3C) to promote the Semantic Web activity [McGuinness and Harmelen, 2004]. OWL is more expressive than the RDF because it provides additional vocabulary with formal semantics for describing properties and classes such as disjointness, cardinality (e.g. exactly one), equality, symmetry, among others. In a nutshell, OWL has three increasingly-expressive sub-languages: *OWL Lite*, *OWL DL*, and *OWL Full*. *OWL Lite* supports those users primarily needing a classification hierarchy and simple constraints. *OWL DL* supports those users who want the maximum expressiveness while retaining computational completeness and decidability. *OWL Full* is meant for users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees. For example, in *OWL Full* a class can be treated simultaneously as a collection of individuals and as an individual in its own right. *OWL Full* allows ontology to augment the meaning of the pre-defined (RDF or OWL) vocabulary [McGuinness and Harmelen, 2004].

In addition to W3C, the official institute which defines the OWL patterns for the Semantic Web community; the *Protégé-OWL Editor* ¹², an extension of

¹⁰ <http://www.ontoknowledge.org/tools/ontoedit.shtml>

¹¹ <http://www.w3.org/TR/owl-features/>

¹² <http://protege.stanford.edu/overview/protege-owl.html>

Protégé ontology editor, contributes enormously for the OWL diffusion among beginners [Noy et al., 2001].

4.6.5. DARPA Agent Markup Language (DAML)

The *DARPA Agent Markup Language*¹³ (DAML) program officially began in August, 2000. The goal of the DAML effort is to develop a language and tools to facilitate the concept of the Semantic Web. The DAML program has generated the DAML+OIL markup language, which contains a defined syntax, layered on RDF and XML, and can be used to describe sets of facts to build ontology. DAML+OIL makes of RDF namespaces to organize and assist with integration of different and incompatible ontologies. Current research into DAML is leading toward the expression of ontologies and rules for reasoning and action. Nowadays, DAML+OIL provides a basic infrastructure that allows a machine to make the same sorts of simple inferences that human beings do [Connolly et al., 2001].

This Section has described the most outstanding *Mark-up Ontology Web Languages* built on top of Semantic Web trends, however, it is recognized that these languages are originated from previous *description logics languages* such as F-Loci, Cyc, Frame, KIF [Gomez-Perez et al., 2004]. In this dissertation, such languages will not be detailed by the fact that they are not in scope of this study and mainly because the *Mark-up Ontology Web Languages* can indirectly show their roles for knowledge representation.

4.7. Outstanding Semantic Web Projects for Knowledge Management

With the dissemination of the Semantic Web, some outstanding projects have been started in the research community interested in explore the benefits provided by the technology. Aware to the Semantic Web contribution at

¹³ <http://www.daml.org/>

international conferences, this Section presents referenced projects in the sphere of knowledge management.

The **Ontolingua**¹⁴ project provides a distributed *collaborative* environment to browse, create and modify ontologies. The Ontolingua server supports over 150 active users, working to the same ontology under broad coordination supervision, extremely helpful for distributed teams which intend work in cooperation [Farquhar et al., 1996]. Ontolingua is an academic project of the Stanford University Knowledge Systems and is financially supported by industrial organizations such as Defense Advanced Research Projects Agency (DARPA), Department of the Navy, Rapid Knowledge Formation (RKF) program and Boeing Corporation.

Jena¹⁵ is Java framework for building Semantic Web applications. It is widely referenced within the semantic web development and research community due to the facilities for ontology handling and reasoning. The Jena Framework includes a RDF API, reading and writing RDF in RDF/XML, N3 and N-Triples, an OWL API, in-memory and persistent storage and the SPARQL query engine. The World Wide Web Consortium (W3C), for example, runs an RDF validation service that checks whether a document is valid RDF/XML, translates the incoming document into an RDF graph, and displays the graph both as a list of triples and as a diagram. The service uses ARP¹⁶, a Jena's parser, to translate the incoming RDF/XML document into a stream of triples [McBride, 2001]. Jena is open source and grown out of work with the HP Labs Semantic Web Programme¹⁷.

Protégé is a free, open source ontology editor and knowledge-base framework that can be adapted to enable conceptual modeling with new and evolving Semantic Web languages [Noy et al., 2001]. The Protégé platform supports two main ways of modeling ontologies via the Protégé-Frames and Protégé-OWL editors. Protégé ontologies can be exported into a variety of formats including RDF(S), OWL, and XML Schema. Figure 2.2 shows the

¹⁴ <http://www.ksl.stanford.edu/software/ontolingua>

¹⁵ <http://jena.sourceforge.net>

¹⁶ <http://www.hpl.hp.com/personal/jjc/arp>

¹⁷ <http://www.hpl.hp.com/semweb>

“*Classes Tab*” where it possible define classes hierarchy, slots and slot-value restrictions, relationships between classes and properties of these relationships.

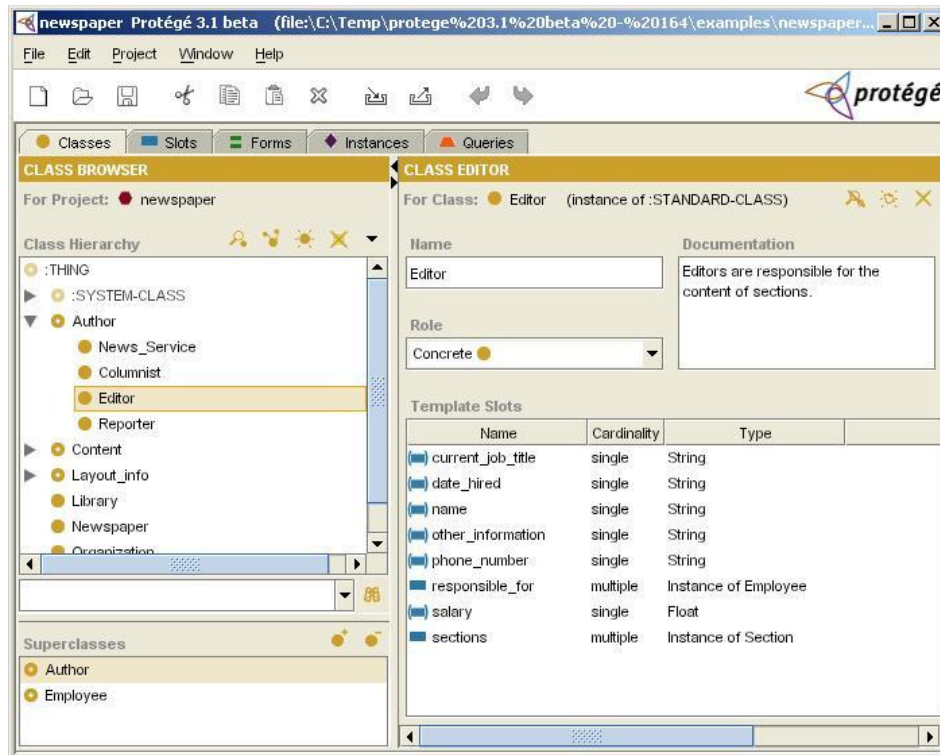


Figure 4.2. Protégé Screenshot.

Protégé was developed by Stanford Center for Biomedical Informatics Research at the Stanford University School of Medicine and yearly promotes its International Protégé Conference where are discussed current and future applications of the tool.

Protégé-OWL Api – An open-source Java library for the Web Ontology Language and RDF Schema. The API provides classes and methods to load and save OWL files, to query and manipulate OWL data models, and to perform reasoning. The great aspect is that Protégé-OWL is tightly integrated with Jena allowing total interactivity between the concurrent semantic web API's. Furthermore, the API is optimized for the implementation of graphical user interfaces [Noy et al., 2001]. Like the Protégé editor, Protégé-OWL Api was developed by Stanford Center for Biomedical Informatics Research at the Stanford University School of Medicine.

The **On-to-Knowledge** ¹⁸ project is exploiting ontologies to develop a methodology and tools for automatically acquiring, maintaining and accessing weakly structured data sources. The project intends to develop a tool suite for efficiently processing large numbers of heterogeneous, distributed and semi structured documents typically found in large company intranets and on the Internet. The initial approach integrates Semantic Web search technology, document exchange via transformation operators, automated information extraction and systematic support for information maintenance and user-specific views. The goal of the project is to provide some knowledge management through guidelines, concepts and tools for enterprises, helping knowledge providers to present their knowledge efficiently and effectively [Fensel, 2002]. Nowadays, all activities of the On-to-knowledge project have migrated to the Digital Enterprise Research Institute ¹⁹ (DERI) whose mission is to establish semantics as a core pillar of modern computer engineering. DERI institute is financially supported by Enterprise Ireland, Information Society Technologies and Science Foundation Ireland.

OntoLearn is a methodology and a battery of software tools that use text mining and statistical techniques to construct domain ontologies for automatic semantic annotation. It uses available resources such as glossaries, document archives, databases, etc., to identify the relevant domain concepts and build formal definitions from informal ones. OntoLearn methodology is expected to be applied during the ontology engineering to facilitate the task of domain specialists who inspect and evaluate the newly acquired domain ontology. The project has been used in national and international projects in several domains, such as tourism, enterprise interoperability, computer networks, and finance [Missikof et al., 2002].

The industrial **OntoClean** project develops a methodology for ontology-driven conceptual analysis that utilizes meta-properties to impose several constraints on the taxonomic structure of the ontology in order to evaluate the choices made. Other problem treated by the OntoClean is the polysemy in which a term may be represented by multiple meanings. For example, "book" is a polysemous term with at least two meanings: a bound volume with a size,

¹⁸ <http://www.ontoknowledge.org>

¹⁹ <http://www.deri.ie>

weight, position, and so forth; an abstract entity with an author, title, and possibly many manifestations [Holsapple and Joshi, 2002]. The OntoClean is financially supported by the Laboratory for Applied Ontology (LOA)²⁰ which performs applied research on the ontological foundations of conceptual modeling, and Ontology Works²¹ that is a product company offering a broad suite of semantic technologies including deductive information repositories.

Simperl et al. [Simperl et al., 2006] has proposed the ***Ontology Cost Model (ONTOCOM)***, a model to predict the costs arising in ontology engineering processes. The work introduces a methodology to generate a cost model adapted to a particular ontology development strategy, and an inventory of cost drivers which influence the amount of effort invested in activities performed during an ontology life cycle. This work has been partially supported by the European Network of Excellence “KnowledgeWeb-Realizing the Semantic Web”, as part of the KnowledgeWeb researcher exchange program T-REX, and by the European project “Sekt-Semantically-Enabled Knowledge Technologies” and “NeOn - Lifecycle Support for Networked Ontologies”.

In the process of building new ontologies, it is common to extending the existing ontologies or combining knowledge from different smaller ontologies. To support the ontology merging, Lambrix and Tan [Lambrix and Tan, 2007] have proposed **KitAMO**: A tool for evaluating ontology alignment strategies. In a nutshell, the tool calculates similarities between the terms from the different ontology sources through linguistic matching strategies. This tool seems to be a promising framework for *OntoLearn* and *OntoClean* methodologies described previously.

As seen previously, the semantic web has called the attention of many areas of software engineering particular to knowledge management. Nowadays, tools and methodologies start to be mature in the way to be profitable business for the organizations. In the next Section, the semantic search engines are in evidence.

²⁰ <http://www.loa-cnr.it>

²¹ <http://www.ontologyworks.com>

4.8. Semantic Search Engines

In addition to methodologies, ontology editors and frameworks, Semantic Search Engines have been developed aiming to enhance their search capabilities. This Section focuses on these tools and how ontologies have been applied to improve their search quality.

Guarino et al. [Guarino et al., 1999] presents the **OntoSeek**, a system designed for content-based information retrieval from online yellow pages and product catalogs. Guarino et al. believe that structured content representations coupled with linguistic ontologies can increase both recall and precision of content-based retrieval. In spite of not focusing on source code retrieval, Guarino's approach focuses on solving semantic-match problems by using linguistic ontologies such as *WordNet* and structured representation formalisms. Similar to OntoSeek, the **Semantic Web Search Engine** (SWSE)²² [Breslin et al., 2005] is a search engine for Semantic Web data that utilizes vocabularies and ontologies to make it possible to apply powerful inference techniques and perform relevant searches. SWSE is a research project being carried out by DERI Galway²³

Swoogle is a crawler-based indexing and retrieval system for Semantic Web documents in RDF or OWL. Swoogle reasons about these documents and their constituent parts (e.g. terms and triples) and records and indexes meaningful metadata about them in its database. It provides services to human users through a browser interface and to software agents via web services. Several techniques are used to rank query results inspired by the *PageRank*²⁴ algorithm developed by Google but adapted to the semantics and use patterns found in semantic web documents [Ding et al., 2004]. Swoogle is a research project being carried out by the ebiquity research group in the Computer Science and Electrical Engineering Department at the University of Maryland, Baltimore County (UMBC). Partial research support was provided by DARPA and by NSF institute.

²² <http://swse.deri.org>

²³ <http://www.deri.ie/teaching/invited-talks/archive>

²⁴ <http://www.google-watch.org/pagerank.html>

Popov et al. [Popov et al., 2004] has developed **KIM**, a system that allows semantic annotation, indexing, and retrieval of documents with respect to real-world entities. For the end-user, a browser plug-in highlights existing entities in KIM ontology and generates a hyperlink used for further exploring the available knowledge to the entity. A semantic query web UI allows specification of a search query that consists of entity type, name, attribute and relation restrictions that are semantically annotated in the KIM index base. In the KIM Knowledge Base are modeled 80.000 entities describing about its specific type, aliases, expressing the (most probable) official name, attributes (e.g. latitude of a Location), and relations. This is an academic project under development at University of Sheffield, Sofia, Bulgaria.

Semantic-Based Approach to Component Retrieval is a system that utilizes domain ontology for retrieving source components, using additional semantic information to process user query. The authors assume the availability of a domain ontology containing terms, relationships and constraints aiding users to specify their requirements when using natural language (nominal or imperative sentences). The ontology is used to expand the query with synonyms to broaden the search, if needed. The search is centralized in a reuse repository strict to an auction specific domain [Sugumaran and Storey, 2003].

Chen et al. [Chen et al., 2007] present an enterprise solution where they extended the traditional keyword search engine mechanism to a ontology-based search. The SRC Company uses a **San Diego SuperComputer's Storage Resource Broker**: tera-byte size data grid which manages distributed data collections such as source codes, documents, models, etc. The SRB search methods involve keyword searching of the available metadata, however, this approach has some limitations as organizations want to share or integrate data sets. This problem lead the company to enhance the system's search capabilities by implementing semantic search engine and interface built on top of an OWL ontology, RDF instance data and a Jena reasoning engine that enables easier and more sophisticated searching of heterogeneous data stored using SRB.

As seen previously, *Mark-up Web Semantic Ontology Languages* have been applied in traditional keyword search engines in order to improve the search capability in spite of being originally they defined to be applied for web

development. Moreover, as discussed, source code search engines such as *Semantic-Based Approach to Component Retrieval* and *SRB semantic searcher* also utilize *Mark-up Web Semantic Ontology Languages* for enrich their search precision and take advantages of the technology.

4.8.1 Semantic Search Engines

The main goal of the semantic assistance is to provide additional knowledge about the user request in which user's keywords are mapped into concepts of the domain ontology which model a rich set of the semantic relationships comprising subsumption, synonymy and business constraints. The use of ontologies opens a range of meaningful around the object of interest which contribute to reduce the chances of poor query construction and consequently irrelevant search returns. The constraints (relationships and properties) of an ontology aggregate value to the search by narrowing the scope of the request. This benefit goes against the ambiguity problems where a single keyword can be used in different senses and may lead search engines, for example, to retrieve information from unsuitable domains. Therefore, combining information retrieval technique with semantic web assistance, they possibly will configure a promising solution for building robust search tools which prime for knowledge support and relevant returns. Motivated by this premise and based on the mentioned related search tools, this dissertation applies Semantic Web resources in a source code search engine for ambiguity treatment in order to improvements of the search precision. The next Chapter depicts the entire implementation of the proposed solution as well as the benefits acquired with the semantic support.

As seen, the use of domain ontologies have been shared among search engines of different proposes. The OntoSeek, Swoogle, KIM and SWSE utilize ontologies for searching web content on the Internet while *Semantic-Based Approach to Component Retrieval*; SRC broker focuses their search over software assets in specific repositories. Independent of the target, they utilize ontology support to enhance the query construction.

4.9. Chapter Summary

In this Chapter, it was discussed how semantic web technologies have evolved in favor of semantics in software applications. It was presented an overview about ontology concepts and aspects of development. In addition, outstanding semantic markup web languages were described as well as ongoing projects developed around the world. Moreover, it was presented a couple of semantic search engines and their benefits achieved when using ontologies to enhance their search capabilities and a discussion how semantic assistance will evolve towards robust information retrieval tools. In the next Chapter, the entire proposal is detailed.



Semantic Search Engine

Information Retrieval is the interdisciplinary science which underlies computer-based text search tools in order to facilitate the information access. As seen in the Chapter 2, there are many different mechanisms to recover information aiming for quality in the retrieval such as context awareness, facet-based and folksonomy. Anchored in the theory studied so far which combines the benefits acquired with semantic web activity, information retrieval techniques and software reuse, a *semantic layer* to incorporate a source code retrieval tool, aiming to increase the precision of the search results, is proposed.

The semantic layer was designed to be plugged-in any search engine, however, this proposal takes place in the B.A.R.T (Basic Asset Retrieval Tool) search engine (see Chapter 3) due the fact that this dissertation is part of the RiSE framework. Moreover, this study makes part of an evolutionary agenda on how different search techniques may assist developers to find reusable software assets more efficiently. The benefits and improvements achieved in the final of this study will be carefully analyzed and hopefully applied in *commercial* releases of the search engine.

This Chapter is organized as follows: Section 5.1 describes the requirements proposed for the semantic search and Section 5.2 presents the system architecture as well as the modules which compose it. Section 5.3 details the semantic core components which implement the semantic layer. Section 5.4 describes the improvements carried out for the proper use of the semantic features. Section 5.5 lists the set of frameworks and technologies utilized during the implementation. Section 5.6 simulates a semantic search in action to make the clear the benefits of the proposal and, finally, provided, Section 5.7 summarizes the content presented in this Chapter.

5.1. Requirements

The requirements (functional and non-functional) proposed for the semantic search engine are based on problems faced by outstanding search engines analyzed so far [Prieto-Díaz, 1991], [Garcia et al., 2006b], [Vanderlei et al., 2007], related work in the literature such as [Henninger, 1994], [Guarino et al., 1999], [Ye and Fischer, 2002], [Sugumaran and Storey, 2003], and reviews of the state-of-the-art [Lucrédio et al., 2004]. Adding to this, reuse practices and discussions in the RiSE group. In this context, a set of functional and non-functional requirements for the semantic layer are presented in the following Subsections.

5.1.1. Functional Requirements

This subsection outlines the functional requirements for the development of the semantic search engine. Initially, the primary requirements for effective search and retrieval are described followed by the semantic requirements.

- **Keyword Search and Retrieval** - The search mechanism should be performed through keywords usage, like most web search engines, in order to avoid the learning of a new method. Thus, the search must accept a string as the input, and must interpret logical operators such as “AND” and “OR”;
- **Query formulation** - There is a natural information loss when the user is formulating a query. As pointed out by [Ye and Fischer, 2002], there is also the conceptual gap between the problem and the solution, since usually components are described in terms of functionality (“how”), and queries are formulated in terms of the problem (“what”). A search engine must provide means to help the user to formulate the queries, consequently reducing this gap;

- **Search results presentation** - The search result must be presented in the developer's environment, so he can more easily reuse the source code into the project that he is currently working on;
- **IDE Integration** - Ideally, the source code search tool should be integrated to the developer's IDE, so that minimum overhead is required in order to use it. A flexible idea is to use plug-in based integration, such as in the Eclipse platform; and
- **Filter Source Code** - Although ideally all kinds of code should be considered for reuse, an automatic mechanism depends on a certain level of quality. In this sense, the search must not retrieve source code with minimal documentation such as less than 30% (number extracted through code metric tools). Thus, a qualitative analysis of the codes must be performed, in order to eliminate low-quality asset that could prejudice the understanding and reuse of the item returned.

The subsequent functional requirements are focused on the semantic features and represent the core functionalities of the proposed solution.

- **Development of a Domain Ontology** – A domain ontology should be created and completed with technological terms handle by source codes. The ontology must identify technological domains and related terms besides determine possible relationships among them. The domain chosen is justified by the fact that nature of the searches focuses on source codes;
- **Management of Context in the Domain Ontology** – The ontology concepts and their relationships stated in the ontology are expected to be universal; however, this is an utopist view since the semantics about some concepts may vary according to regions and habits. In this sense, the ontology must support the management of the context for the concepts described in the ontology;
- **Semantic Possibility Browsing** - The inferred domains from the ontology reasoning must be shown in the developer working set in order to help the query contextualization. The semantic browsing must be performed together with the returns of keyword query;

- **Search by Semantic Possibilities** - The search by source codes must take into account the semantic possibility chosen by the developer. The search engine must tackle the additional to retrieve codes into the domain expected;
- **Semantic Query Formulation** – The search mechanism must expand the keyword query as soon as the semantic domain is chosen. The final query is formulated by the composition of the keyword input and the related technical terms from semantic domain;
- **Source Code Classification and Semantic Indexing** – The source codes of the reuse repository should be analyzed and classified into a proper domain category. This classification must be utilized to compose the index structure during the index process. The semantic indexing is required to maximize the performance of the information retrieval mechanism by focusing the search on classes of a given domain; and
- **Visualization of the Source Code Classification** – In general, the effect of the source code classification is only perceived when semantic search is well succeeded; therefore, in order to turn the classification more visible for the system administrators, a graphic visualization of the categorization might be useful for anticipating validation of the classification.

In the following subsection the non-functional requirements are described.

5.1.2. Non-Functional Requirements

The non-functional requirements are: reusability, extensibility, usability, high precision and recall and interface with other search techniques.

- **Reusability** - the proposed semantic layer must be built under the *Component-Based Development* in order to promote the reuse activity by producing self-contained components instead of an integrated application. Moreover, future applications which desire to reuse such features only will plug such components in their environment;

- **Extensibility** - In general, any software application must take into account future growth. The architecture constraints must presume the addition of new functionalities and the level of effort required to implement them without impacting to existing system functions. For this reason, the proposed solution must be well-structured, with low coupling modules to accommodate exactly maintenance and extension demanded. This is a high priority requirement;
- **Ubiquity** - The proposed solution must provide the possibility of being set up with existing environments and tools at the same time without reducing its performance;
- **Usability** - Software application must be concerned with human interaction due to the easiness of operating software implies on its popularity and attractiveness. Therefore, the graphic interface of the tool must be built-in with intuitive components to perform the functionalities. This is a high priority requirement;
- **High Precision and Recall** - This is an essential requirement that must be regarded in any search mechanism. The performance of an information retrieval tool is measured taking into account the achieved precision and recall rates. High precision is achieved when the most relevant elements are returned in a search and high recall is achieved when few relevant elements are left behind. Considering that irrelevant information contributes for reducing the reuse activity;
- **Interface to others search techniques** - The new semantic development must be flexible to be combined with other types of search techniques like keyword matching, facet-based, and folksonomy technique. Integration with others search techniques can be regarded as important requirement once leads the proposal to be extensible;
- **Performance** - Performance is usually measured in terms of the response time. In centralized systems, the involved variables are the hardware processing power and the search algorithm complexity. In distributed scenarios, however, other variables must be considered, such as network traffic, geographical distance and the greater number of components;

- **Platform Independence** – Organizations usually have heterogeneous development platforms and, for that reason, an integrated reuse environment must seamlessly integrate with all existing configurations in order to maximize its user base and consequently provide more effective results. The implementation of the environment functionalities must be based on technologies that are easily portable across existing platforms;

Although the mentioned requirements are considered important for the development of this proposal, some of them were not entirely or partially accomplished due to time constraints and scope of the proposal. In this sense, some requirements had more priority than others. Table 5.1 shows the expected requirements against its situation of development according to its priority, thus, in order to formalize the situation of each one, some priority criteria were adopted:

- **Essential** – It represents the indispensable and high-priority requirements that must be carried out. The lack of them turns the application useless;
- **Important** – It represents the medium-priority requirements that are strongly advisable for better usage of the tool; and
- **Aimed** - It represents the low-priority requirements that are required for particular situations or enhancements for current development.

For the *situation of realization*, three criteria were adopted:

- **Achieved** – It means that the requirement was completely carried out and tested;
- **Partially Achieved** – It means that the requirement was implemented but there was not opportunity for testing or validation; and
- **Not Achieved** - It represents the requirements that were not definitely implemented.

Table 5.1 summarizes the requirements showing the priority and the situation of the proposed requirements.

Table 5.1. Summary of requirements

Requirement	Priority	Situation
Keyword Search and Retrieval	Essential	Achieved
Query formulation	Essential	Achieved
Search results presentation	Essential	Achieved
IDE Integration	Important	Achieved
Source Code Filtering	Aimed	Not Achieved
Development of a domain ontology	Essential	Achieved
Management of Context in the Domain Ontology	Aimed	Not Achieved
Semantic Possibility Browsing	Essential	Achieved
Search by Semantic Possibilities	Essential	Achieved
Semantic Query Formulation	Important	Achieved
Source Code Classification	Essential	Achieved
Visualization of the Source Code Classification	Aimed	Achieved
Reusability	Important	Achieved
Extensibility	Important	Partially Achieved
Ubiquity	Important	Not Achieved
Usability	Important	Partially Achieved
High Precision and Recall	Essential	Partially Achieved
Interface with others search techniques	Aimed	Achieved
Performance	Important	Partially Achieved
Platform Independence	Important	Achieved

Once presented the requirements for creation of the semantic layer, the system architecture with the major modules which compose the B.A.R.T search engine is presented in the next section.

5.2. System Architecture

The B.A.R.T. architecture was designed to be extensible providing the capacity to add new features by including new components. In a nutshell, the B.A.R.T. project is based on client-server architecture where the clients are represented by IDE plug-ins and a Web Interface and the server is represented by business modules responsible for managing the search and retrieval functionalities. The semantic layer will be placed in the current B.A.R.T architecture in accordance with the proposed requirements without comprising the existing development.

5.2.1. B.A.R.T Client

The front-end of the B.A.R.T search engine varies according to the user environment. In this context, in addition to a common Web interface, the B.A.R.T front-end includes an Eclipse and Ms/Word plug-ins which focuses their search on codes and documents respectively. The objective is to put the B.A.R.T capabilities into the user environment taking into account its specific particularities.

Particularly to this dissertation, the semantic proposal will only affect the Eclipse plug-in because it genuinely handles operations over source codes. Nevertheless, it is absolutely guaranteed its deployment on the Web interface and other front-ends employed in source code retrieval.

The bridge between the client and server sides is managed through a communication layer that is implemented by Web Services. This implementation strategy allows the B.A.R.T services to be available anywhere on the Internet, or even on corporative Intranet, in scenarios where the components are proprietary.

5.2.2. B.A.R.T Server

The B.A.R.T architecture is composed by business modules that, in general, perform search and retrieval tasks. The architecture is composed by the following modules: **searcher**, **reasoner** (*new*), **retriever**, **analyzer** (*new*), **indexer**, **filter** and **repository manager**. The semantic layer is represented by the **analyzer** and **reasoner** modules. Figure 5.1 shows the current B.A.R.T. architecture and after a brief description of each module.

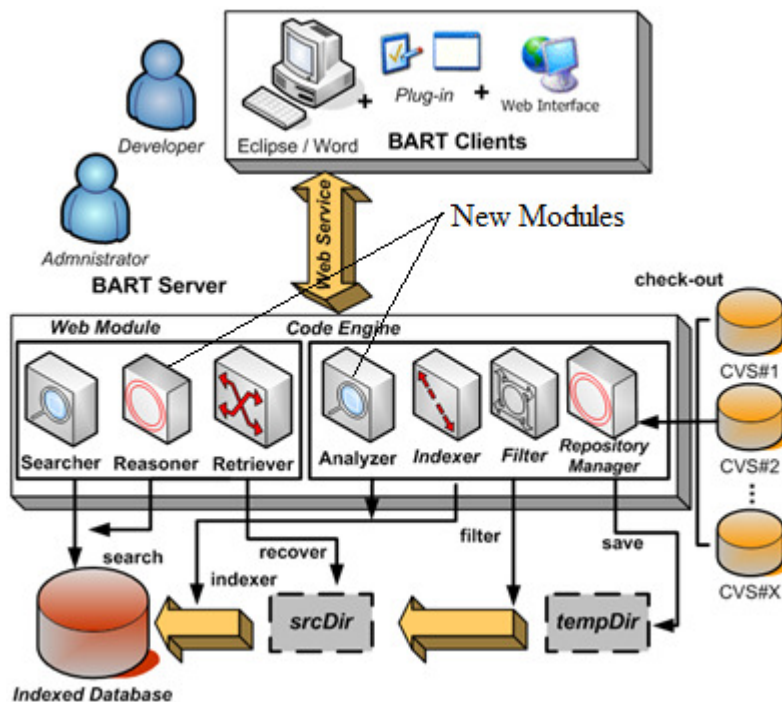


Figure 5.1. B.A.R.T. Architecture

- **Repository Manager** – this module checkouts assets from *Concurrent Version System (CVS)* repositories on the Internet and maintains the reuse repository of the B.A.R.T system. It manages the whole infrastructure for storing the repository assets as well as the index base. Moreover, through this module it is possible to schedule updates from previous checkouts in order to capture brand new assets;
- **Filter** – this module filters those assets which do not satisfy some constraints such as unnecessary extension and lack of documentation (particularly applied to source codes);

- **Analyzer** – It is a brand new module that compounds the *semantic layer*. This module feeds the indexer one with the semantic classification from the source code analysis. It encapsulates the *Semantic Code Analyzer* component detailed in Section 5.4. Nowadays, only source code analysis is supported, however, in the future releases, this feature will be extended to documents written in natural language;
- **Indexer** - This module indexes the reuse repository and creates the index base essential for the search execution. It works in conjunction with analyzer module when performing the semantic indexing;
- **Reasoner** – It is also a new module that compound of the *semantic layer*. In this module, all the ontology management is processed as well as the tasks associated with reasoning and inference. The reasoner module is responsible for providing the domain terms related with a given query in order to help end users to contextualize its keyword query. It encapsulates the *Semantic Query Reasoner* component detailed in the Section 5.4;
- **Searcher** - This module is responsible for processing the user query and process the search over the index base. It works in conjunction with reasoner module when performing the semantic search; and
- **Retriever** - After a successful search, the user has the possibility of downloading the assets browsed in the B.A.R.T client application. This module is responsible for transferring them to the user directory.

To effectively build the semantic layer in the B.A.R.T search engine, the semantic modules had to be integrated with existing development in order to transparently provide users additional resources to maximize the reuse activity. Both **reasoner** and **analyzer** modules were included in the architecture with low coupling without cause side-effect to existing development.

In the server side, the new modules had to be incorporated with minimum impact in the previous architecture without decreasing the system performance. The searcher module was slightly updated to understand the semantic information attached to the keyword query without compromising ordinary queries. Equally, the indexer module has suffered vaguely modifications by adding a domain field in the index structure without

compromising the regular indexing process as well. In the front-end, only the Eclipse plug-in had the graphical interface updated to make visible the semantic assistance during query construction (see 5.3.2).

In this Section, a higher level of abstraction about the proposed solution was presented to provide an architectural overview of the system, however, in the next Section the semantic components that implement the reasoner and analyzer modules are detailed under a more technical view.

5.3. The Semantic Components

A general definition of the architecture of the proposed solution was presented in the last subsection. Moreover, it was possible to visualize how the **analyzer** and **reasoner** communicate with exiting modules in order to satisfy the set of requirements defined in first Section.

As previously stated, both modules have their functionalities implemented by two components respectively: **Semantic Code Analyzer** and **Semantic Query Reasoner**. The first is engaged of creating the semantic indexes while the second is instanced for contextualizing the user query into a proper domain. The following Subsection details both components as well as discusses how they can evolve to deal with other asset types beyond source codes.

5.3.1. Semantic Code Analyzer

The Semantic Analyzer component answers for source code analysis, domain classification and semantic annotations. Basically, it classifies source code in infrastructure domains and annotates such information in the index structure to provide additional information about the code purpose. Such information helps search mechanisms to retrieve codes taking into account their area of application and not merely by keyword matches. As a consequence, it is expected that codes which share the same keywords but belong to distinct domains are *not* retrieved during the semantic search.

Although the main task of the component is classifying source codes into infrastructure categories, it performs prior tasks for augmenting the analysis efficiency such as code filtering and cleaning. Therefore, to characterize the component as a single classifier is not the most adequate denomination. Through the *filtering* functionality, the *Semantic Code Analyzer* limits the classification for the files types exclusively determined by user, ignoring other distinct extensions. Through the *cleaning* functionality, the filtered files have the comments cleaned in order to improve the classification accuracy. Although comments are used for giving contextual information about the program behavior, it is speculated they may puzzle the categorization since no control is assured about the text content.

All this process is performed before the indexing because it is the entry point to fill in the index structure with the additional semantic information. The component responsible for indexing receives the outcome from the semantic analysis and tag the indexes with the classification achieved. The final result is an index base semantically built.

For convenience, the source code analysis is expected to run in a background activity, considering that the size of repository may require some machine effort. On the other hand, for this component, performance is a requirement less important than the classification accuracy.

For categorization, the component has instanced a naive Bayes classifier, a probabilistic model with strong (naive) independence of assumptions [Lam et al., 1999]. Its use is justified by the fact the method is usually applicable for unstructured text documents, including to this: the source code. In spite of being using a naive Bayes probabilistic model, the existence (so far not experimented) of other probabilistic models are recognized such as Supported Vector Machines, Decision Tree, Best Neighbor among others as discussed in Chapter 3. The component architecture was designed to allow easy modification in the algorithm utilized; therefore, the substitution of the categorization method does not require too much effort [Atkinson et al., 1989].

5.3.1.1. Implementation Aspects

According to the B.A.R.T architecture (see Section 5.2) the *indexer module* communicates with the *analyzer module*. In a component view this relationship is represented by the communication between the *Indexer Component* and the *Semantic Code Analyzer* (Figure 5.2).

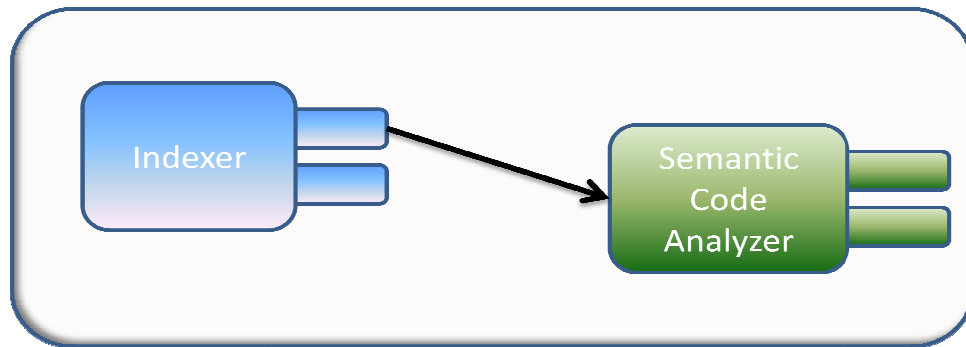


Figure 5.2. Component communication

The *Semantic Code Analyzer* provides two main interfaces in which other components/applications can be plugged such as:

- **Parameter Interface** - For running properly, the component must set up some parameters such as *knowledge base path*, *repository path* and *language type*. This parameter makes the component entirely configurable allowing users specify their own preferences during classification; and
- **Semantic Analysis Interface** - Once the component parameters are defined it is ready to execute the semantic analysis. The external application which reuse this component just need then to make a call for `startSemanticAnalysis()` method.

Internally, the core architecture was thought to be loosely coupled where the business modules may change without affecting or requiring change in any other part of the system. In order to achieve this requirement, the core classes are instanced though *late binding* pattern in which no binding between the

operation names and implementation classes is established at compile time [Atkinson et al., 1989].

5.3.1.2. Source Code Analysis and Classification

For classifying text documents using probabilistic classifiers makes necessary the existence of a knowledge base (or training data) used as reference during the categorization [Rennie, 2001]. As the Semantic Code Analyzer is employed to classify source codes, the knowledge base (KB) with the reference data was assembled with source codes whose scope enclosed technological domains representing the categories (Figure 5.3).

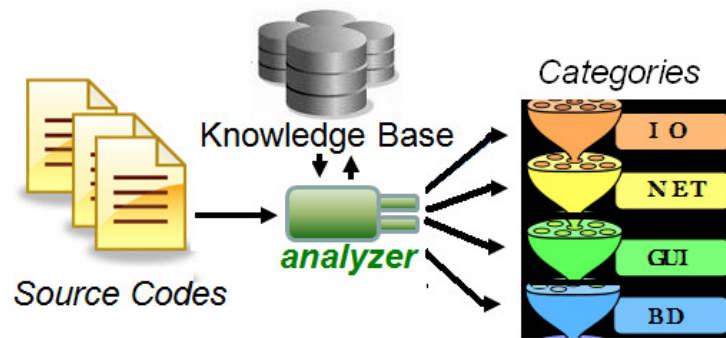


Figure 5.3. Source Code Classification

Exclusively for this study, the domain categories chosen were: ***xml***, ***database***, ***security***, ***GUI***, ***network***, ***math*** and ***i/o***. These categories were chosen by the fact that they are cross platforms domains used in a wide range of applications and consequently may be found in the greater part of the repository. Despite of being working only with seven domains, others may be added on demand with punctual changes in current development. Nevertheless, the limited number of domains is regarded enough to confirm the classification usefulness.

For building the knowledge base, 15 reference codes for each domain (category) have been selected approximately. In the current stage, this process is done manually; on the other hand it can be done automatically by means of intelligent agents or web crawlers for example. This functionality, however, makes part of our negative scope (see Section 5.7) and will not be implemented in the current release. Although the knowledge base is formed with hundreds of

reference files, the Semantic Code Analyzer is able to compact it into a single file providing, therefore, portability on its deployment and reuse.

The classification accuracy depends on the knowledge base quality; therefore, it is strongly advisable that the persons involved on this activity must be familiar with program languages and codification. Choosing the best codes configures the key point for having an efficient knowledge base. Figure 5.4 shows an example of a feasible source code to compose the knowledge base specific to the database domain.

```
1  package org.visres.util;
2
3  import java.sql.*;
4  import java.sql.Date;
5  import java.math.*;
6  import java.util.*;
7
8  public final class SQL {
9
10     private SQL() {}
11
12
13     public static int update(Connection con, String sql) throws SQLException {
14         Statement stmt = null;
15         try {
16             stmt = con.createStatement();
17             return stmt.executeUpdate(sql);
18         } finally {
19             if (stmt != null) stmt.close();
20         }
21     }
22
23
24
25     public static int update(Connection con, String sql, int expected)
26     throws SQLException {
27         Statement stmt = null;
28         try {
29             stmt = con.createStatement();
30             int result = stmt.executeUpdate(sql);
31             if (result != expected)
32                 throw new AssertionError("Value returned (" + result + ") "+
33                                         "was not what was expected "+
34                                         "(" + expected + ")");
35             return result;
36         }
37     }
38 }
```

Figure 5.4. A feasible Java code for database domain

Figure 5.4 shows a code whose content presents methods from “*java.sql*” (line 3) library that throws the “*SQLException*” (line 13) besides the existing keywords related with database domain such as “*sql*”, “*update*” and

“*connection*” (line 25). Source codes like this with such clear evidences of the domain are expected in the knowledge base. Thus, considering that this piece of code belongs to the knowledge base, if an ordinary source code to be classified contains such methods, imports or attributes, then, its probability will be categorized in the **database** domain.

In this current version, the knowledge base is being populated manually without any automatic support; however, for future versions it is quite advisable the existence of graphical user interface where experts could contribute with the knowledge base improvement. The profile of the contributors is another point that must be concerned with so that this impacts directly in the classification accuracy. It is prudent that such activity could be performed by software developers or people able to effectively identify the correct code domain and mainly distinguish which code can be elected to compound the knowledge base.

5.3.1.3. Semantic Indexing

In general, the main contribution of this component is to provide the domain classification about the codes that will be indexed. Such information will be used as another filter option when retrieving codes by domain. For achieving this, the index structure had to be updated with the new metadata field. Figure 5.5 shows the index structure with the domain field.

<contents>	ITS-----	1.0	package aosdplugin;\r\n
<domain>	ITS-----	1.0	api.gui ←
<fileName>	--S-----	1.0	FrameDirectories.java
<filePath>	I-S-----	1.0	C:\Documents and Settings\if

Figure 5.5. The metadata domain in the index structure

Figure 5.5 shows the fields which compose the index structure; besides the contents, filename, file path, loc, module and repository fields, it was added the domain field to accommodate the semantic classification of each source code analyzed. In order to turn the classification more visible for users, it was provided a graphic visualization of the codes categorized. Figure 5.6 shows the “i/o” domain with its respective codes.

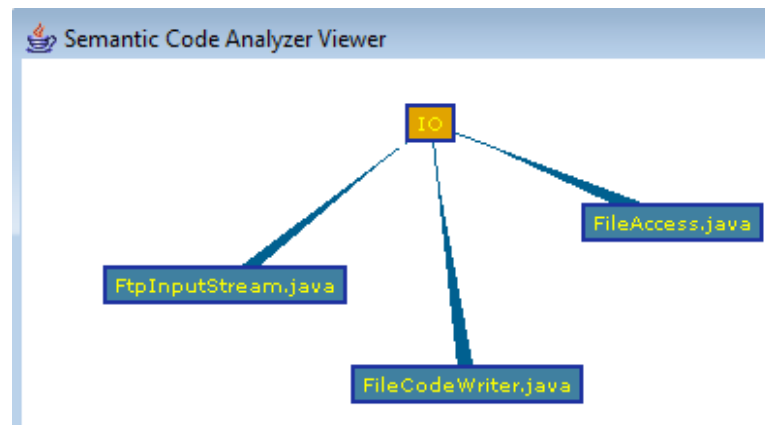


Figure 5.6. Semantic Code Analyze Viewer

The *Semantic Code Analyze Viewer* was placed in the B.A.R.T Server to allow administrators have a graphical view of the result of the source code classification. According to Figure 5.6, the source codes *FtpInputStream.java*, *FileCodeWriter.java* and *FileAccess.java* were classified in the **input/output (IO)** domain. This feature also presents a significant limitation that is not covered in this version: the visualization is harmed when number of *classes per domain* is higher than 200. It is speculated that the change of the framework which provides the exhibition could be a feasible solution.

5.3.2. Semantic Query Reasoner

The *Semantic Query Reasoner* is the component responsible for enhancing the user query with related terms in order to match more relevant source codes. The objective is to contextualize the keyword (query) in a specific domain in order to avoid ambiguity in the search return. In consequence, it is expected to increase the search efficiency since it will be focused on a set of codes instead of entire repository.

In order to provide the appropriate related terms, the component reasons over a domain *ontology* while the ordinary keyword search is performed. As a consequence, in addition to the codes returned, related domain terms are suggested for placing the query into a specific context. Once a domain term is chosen, the search is focused on codes which belong to the domain selected. This initiative reflects Henninger's claim [Henninger, 1994], which state that

constructing queries is as important as (or more than) the retrieval algorithm used. The *Semantic Query Reasoner* required some improvements in the Eclipse B.A.R.T plug-in. Figure 7 shows the Eclipse IDE environment, with the B.A.R.T plug-in on the right side and the semantic outcome highlighted.

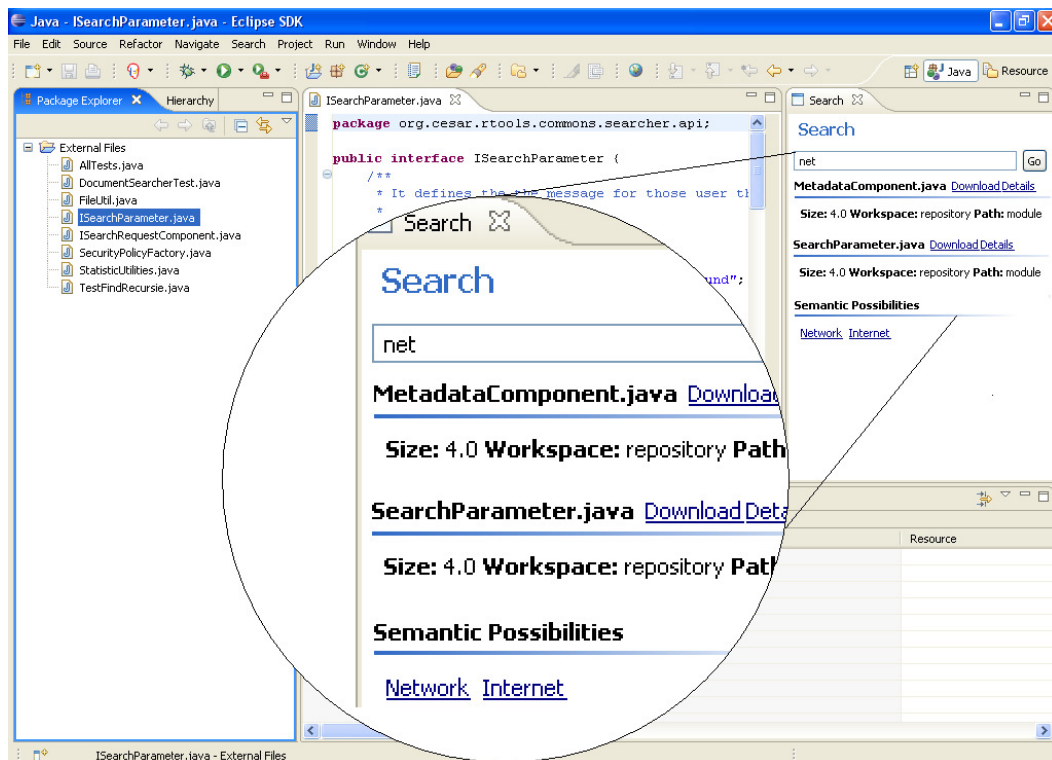


Figure 5.7. Semantic Possibilities at Eclipse Plug-in

According to Figure 5.7, in addition to the query field and the code result frame, the *semantic possibility view* was added. The *semantic possibilities* (or domain terms) are provided from a domain ontology that maps possible user queries in technical terms. Thus, it is expected to bridge the user's need in technological terms handled by source codes to avoid the semantic conceptual gap in the query formulation. Historically ontologies have been employed to achieve better precision and recall in text retrieval systems since they may precisely describe collections of concepts and their interrelationships regarding a specific domain (Chapter 4). Consequently, this benefit naturally avoids possible anomalies such as synonymy and polysemy. According to Golder and Huberman [Golder and Huberman, 2006] synonymy happens when two words have the same meaning while polysemy occurs when the same words have

different meanings. In the following Subsection, the component architecture is showed.

5.3.2.1. Implementation Aspects

According to the B.A.R.T architecture (see Section 5.2.2) the reasoner module communicates with searcher module. In a component view, this relationship is represented by the communication between the *Semantic Query Reasoner* and the *Searcher Component* (Figure 5.8).

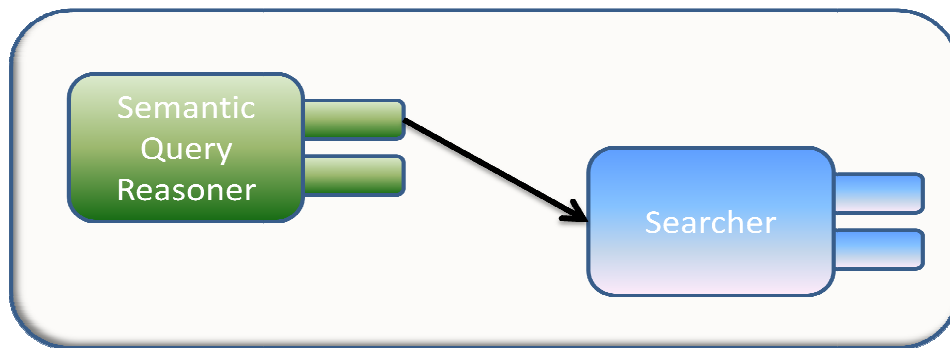


Figure 5.8. Component communication

The *Semantic Query Reasoner* provides two main interfaces in which other components/applications can be plugged:

- **Parameter Interface** - For running properly, the component must set up some parameters such as *ontology path*. This parameter makes the component entirely configurable allowing users specify their own preferences for specialized reasoning; and
- **Semantic Inference Interface** - Once the component parameters are defined it is ready to execute the semantic reasoning. The external application which reuse this component just need then to make a call for `startSemanticInference()` method.

Similar to the Semantic Code Analyzer, this component architecture was loosely coupled built and have used late binding pattern. In the following

Subsection, the ontology building process model as well as the ontology constraints is depicted.

5.3.2.2. The Ontology Model

In the process of ontology building, an intense Semantic Web commitment had to be employed in order to establish the ontology vocabulary. In accordance with the search focus, the terms had to comprise a technical terminology encountered in source code or associated with it. This motivated the investigation of similar ontology models on the Internet through existing semantic web search engines such as Swoogle²⁵. This phase was important for reusing previous knowledge already raised in other ontologies so that it could collaborate to enhance and validate the proposed model. The objective was to follow the ontological engineering process described by Guarino [Guarino, 1998] and Gruber [Gruber, 2002], which encourages systematic development by knowledge reuse.

The ontology model was created using the *Ontology Web Language* (OWL), an expressive *Mark-up Ontology Web Language* highly diffused among the Semantic Web community and broadly utilized among Semantic web applications as seen in Chapter 3. The current ontology is available at <http://cin.ufpe.br/~fad2/infra.owl> and contains 130 classes, 640 individuals and 2 *DataType* properties: *hasKeywordAssociated* and *hasInfraTerm*. This numbers of the ontology model are based on last update on 6th January, 2008, therefore, slight variation may be evidenced in future dates. Figure 5.9 shows some ontology classes (representing the domains), e.g., “*Security*” and its subclasses “*Authentication*”, “*Authorization*” and “*Cryptography*” in the Protégé Editor.

²⁵ <http://swoogle.umbc.edu>

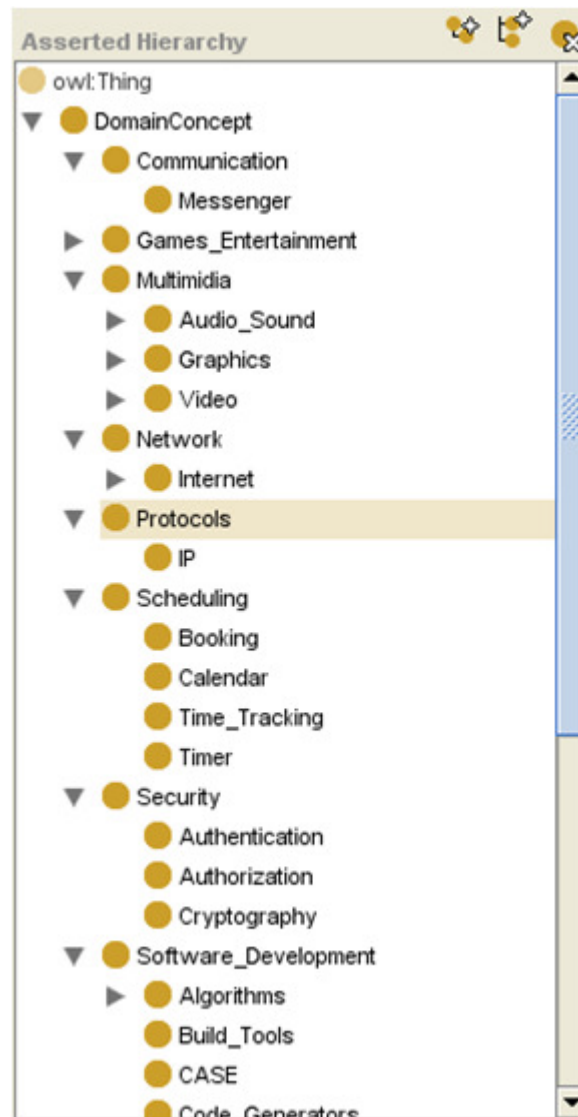


Figure 5.9. Screenshot of the Ontology Model

As previously said, the ontology is also composed with two *DataType* properties: “*hasKeywordAssociated*” which associates ontology classes (representing the technical domains) with keywords and the “*hasInfraTerm*” property that associates ontology classes with infra terms:

- **hasKeywordAssociated** - This property associates ontology classes with the most feasible terms that could be used as keyword in the query. Thus, it is possible to state the following relation: “Security *hasKeywordAssociated* authentication” considering “Security” as the Subject, “*hasKeywordAssociated*” as the property and “authentication” as the literal value. As a consequence, if the user query is

“*authentication*”, then likely “*Security*” will be returned as one of the available semantic possibilities to augment the efficiency of the query.

- **hasInfraTerm** - This property associates ontology classes with the most common infrastructure terms of a given domain. This allowed the following relation to be established: “*Security hasInfraTerm JAAS*” considering “*Security*” as the Subject, “*hasInfraTerm*” as the property and “*JAAS*” as the literal value. The “*JAAS*” technology corresponds to the set of APIs that enable services to authenticate and enforce access controls upon users. Through this “*hasInfraTerm*” property, if the user chooses the “*Security*” domain, among the semantic possibilities available, the query will be enhanced with the “*JAAS*” term.

Figure 5.10 shows a part of the generated OWL document that associates “*Cryptography*” and “*Math*” domains with their respective keyword terms.

```
<Cryptography rdf:ID="_Cryptography">
  <hasKeywordAssociated rdf:datatype="string">algorithm</hasKeywordAssociated>
  <hasKeywordAssociated rdf:datatype="string">codigo</hasKeywordAssociated>
  <hasKeywordAssociated rdf:datatype="string">cryptography</hasKeywordAssociated>
  <hasKeywordAssociated rdf:datatype="string">algoritmo</hasKeywordAssociated>
  <hasKeywordAssociated rdf:datatype="string">encode</hasKeywordAssociated>
  <hasKeywordAssociated rdf:datatype="string">criptografia</hasKeywordAssociated>
</Cryptography>
<Math rdf:ID="_Math">
  <hasKeywordAssociated rdf:datatype="string">multiply</hasKeywordAssociated>
  <hasKeywordAssociated rdf:datatype="string">distance</hasKeywordAssociated>
  <hasKeywordAssociated rdf:datatype="string">max</hasKeywordAssociated>
  <hasKeywordAssociated rdf:datatype="string">high</hasKeywordAssociated>
  <hasKeywordAssociated rdf:datatype="string">integer</hasKeywordAssociated>
  <hasKeywordAssociated rdf:datatype="string">rounding</hasKeywordAssociated>
  <hasKeywordAssociated rdf:datatype="string">bigDecimal</hasKeywordAssociated>
  <hasKeywordAssociated rdf:datatype="string">magnitude</hasKeywordAssociated>
```

Figure 5.10. Screenshot of the OWL document

Given that the ontology model acquired a satisfactory taxonomy, the design of the ontological queries was started in order to tackle the OWL model and retrieve the semantic possibilities. The queries were constructed under SPARQL²⁶ syntax anchored in the fact that too much and easy support is provided on the web besides being broadly applied among semantic web applications. Two initial queries were developed: *Domain Query* and *Infra Query*, one for each *DataType* property:

²⁶ <http://www.w3.org/TR/rdf-sparql-query/>

- **domainQuery** – this query is employed for seeking domains terms related to the user’s keyword through the property *hasKeywordAssociated*. This query reveals possible polysemy anomalies, since it come up with different domains related to the same keyword; and
- **infraQuery** – it is a complementary query of *domainQuery* so that it retrieves infra terms related to a given domain. Basically, this query makes use of the property “*hasInfraTerm*” to retrieve technical terms encountered in an associated API specification.

In parallel to *domainQuery*, rules were applied for identifying distinct domains that eventually could satisfy a particular proposition. The objective was to retrieve domains initially isolated but frequently evidenced during the code development. As an example, codes originally from the *GUI* domain usually require Math functionalities. Thus, if the *GUI* domain is returned to compound the semantic possibilities, the *Math* domain must be retrieved as well according to the rule. In this way, it is intended to provide associated domains in which the user may need requisite in further searches. Although not many rules have been written, at least the necessary infrastructure to this was implemented. The creation of new rules is envisioned to be happened in an automatic way by means of *association rules*: a data mining technique employed to extract patterns by usage. This interface with data mining technique makes part of the future works and can be deeper seen in the Chapter 7.

Although the population depends on formal approval of experts which manage the ontology, an interface for user interaction was provided. In the B.A.R.T Eclipse plug-in a *Term View* tab was provided in which users could suggest keywords or technical terms of a given domain (Figure 5.11).

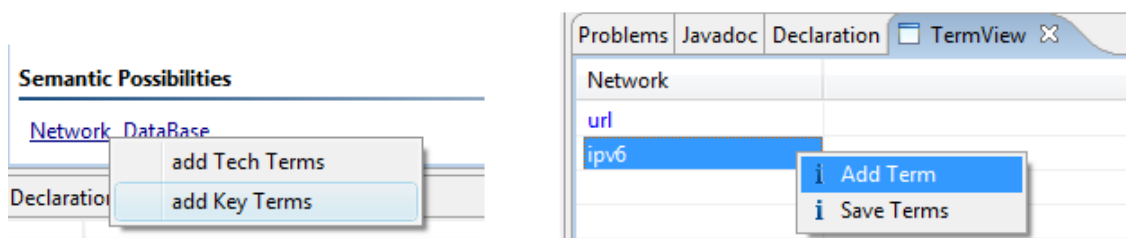


Figure 5.11. Term View tab

In current development, the terms are only saved in local user machine and sent to the B.A.R.T Server. The objective is that the ontology manager could evaluate the suggestions and decide if such terms will compound the ontology vocabulary or not. Nevertheless, this process deserves further discussion and may vary according to the project business. The objective of this feature was to provide an extra channel of populating the ontology, but the decision about what to do with the contributions remain opens.

In the next Section, improvements in the B.A.R.T Configuration Window are described.

5.4. Semantic Configuration at B.A.R.T Eclipse Plug-in

Once the components were integrated in the B.A.R.T search engine, additional improvements in the front-side were made in order to make the semantic assistance configurable. The B.A.R.T configuration window used to allow users to specify the B.A.R.T Server location and authentication inputs. With the semantic features new configurations were added to B.A.R.T configuration window (Figure 5.12).

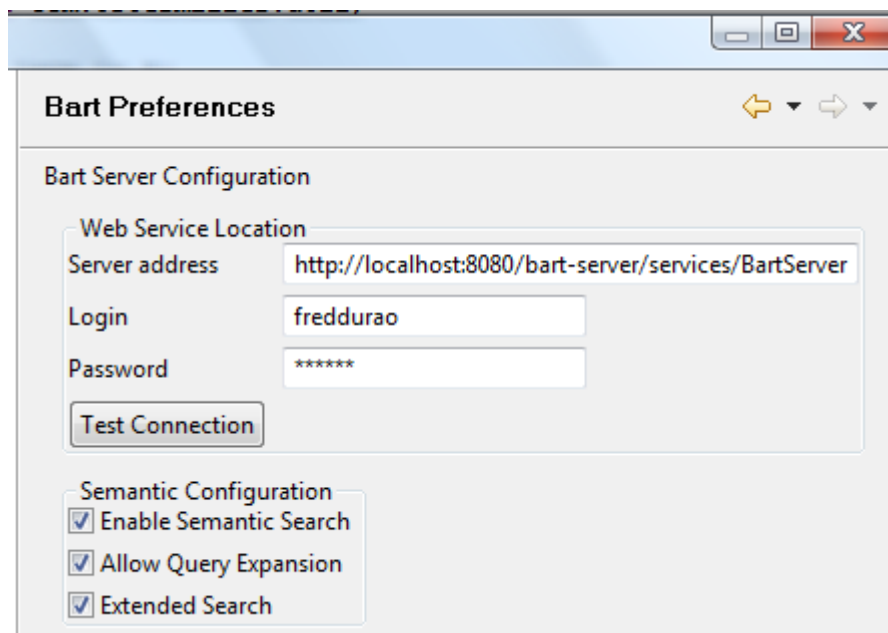


Figure 5.12. B.A.R.T Configuration Screen

Through the “Enable Semantic Search” checkbox, the user may active or not the semantic assistance; when turned off the semantic possibilities are not showed in the B.A.R.T Search perspective. Through the “Allow Query Expansion” option the user may configure the semantic search to be expanded with other related terms or keep only with the current keyword; through the “Extended Search” option the user may configure the semantic search to reach D.N.I files or not.

By default all of the options are checked to lead the user to take advantage of all functionalities provided by the semantic search, however, to provide them configurable goes in favor to the usability requirement.

5.5. Reused Frameworks

The modules which comprise the B.A.R.T architecture have used open-source components to achieve the functionalities desired. This Section outlines the frameworks and libraries used to each module in particular.

The searcher and indexer modules perform their tasks using the *Lucene*²⁷ search engine – a high-performance, full-featured text search engine library maintained by the Apache Group; to checkout source codes from remote repositories, the CVS Checkout Task from Apache *Ant Framework*²⁸ was utilized. The Ant Framework is a Java-based build tool which encompasses a set of projects entailed in resolving a wide range of tasks such as generation of build, documentation, report besides giving support to automate tasks such as unit tests, checkout, validation, etc.

The *Jena*²⁹ framework was utilized in the reasoner module for rule inference and for whole semantic operations. Jena is a framework for ontology handling commonly utilized in semantic web applications supported by HP Labs where. Though Jena framework is possible to load ontology, access its properties, list classes and individuals, fire rules, write and update an ontology

²⁷ <http://lucene.apache.org>

²⁸ <http://ant.apache.org>

²⁹ <http://jena.sourceforge.net>

model. To build the ontology, the Protégé 3.2 ³⁰ ontology editor was utilized due the facilities offered as for visualization as for manipulation. In means of 2006, Protégé has released the Protégé-Owl API ³¹ - a competitor of Jena framework - which provides classes and methods to load and save OWL files, to query and manipulate OWL data models, and to perform reasoning. This library could be used instead of Jena framework, however, when this dissertation stated to be implementation more information about the Jena utilization was available and this favored the choice of Jena. Today Protégé-Owl API has gained many of fellows mainly because of the Protégé marketing and because this API has worked in cover gaps unresolved in Jena API.

The analyzer module uses the *LingPipe Framework* ³²: a suite of Java libraries for the linguistic analysis and subsequent text classification. In current approach, this framework was applied to categorize source code but this framework could be used to classify texts written in natural language as well. This flexibility permits that the Semantic Code Analyzer component to be extended to other types of software assets such as use case documents for example. The *jSVM (Java Support Vector Machine)* is an alternative classifier employed also in text categorization which utilizes the SVM technique. During the investigation about the technologies to be used, the *jSVM* presented unsatisfactory documentation and lack of support about its correct usage. Nowadays, the project has no official web site and seems to be discontinued.

For displaying the semantic annotation in the index base, the *TouchGraph* ³³ framework was utilized. *TouchGraph* provides a set of interfaces for graph visualization using force-based layout and focus + context techniques. In addition the open-source version, Google has sponsored the commercial activity of the framework. Competitors of this framework are *Jung* ³⁴(*Java Universal Network/Graph Framework*) and *Prefuse*³⁵, both were tested and could utilized instead. The preference for *TouchGraph* was done due the fact that this was ranked easier to be handled with better documentation besides the Google approval.

³⁰ <http://protege.stanford.edu>

³¹ <http://protege.stanford.edu/plugins/owl/api>

³² <http://www.alias-i.com/lingpipe>

³³ <http://www.touchgraph.com>

³⁴ <http://jung.sourceforge.net>

³⁵ <http://prefuse.org>

In the front-end, the semantic improvements in the Eclipse B.A.R.T plug-in were made by use of the *Standard Widget Toolkit (SWT)* ³⁶ and *JFace* ³⁷ libraries, both are rich client Eclipse projects for GUI development and had been utilized even before than semantic contribution. Further details about the frameworks usage will not be found in this dissertation from the fact that such information can be better detailed in the respective web sites.

5.6. Semantic Search In Action

In order to facilitate understanding of the proposed solution, this Section simulates the use of B.A.R.T search engine with the extended semantic features. Before starting, however, the reuse repository must have already been semantically indexed by the *Semantic Code Analyzer* as seen in Figure 5.13.

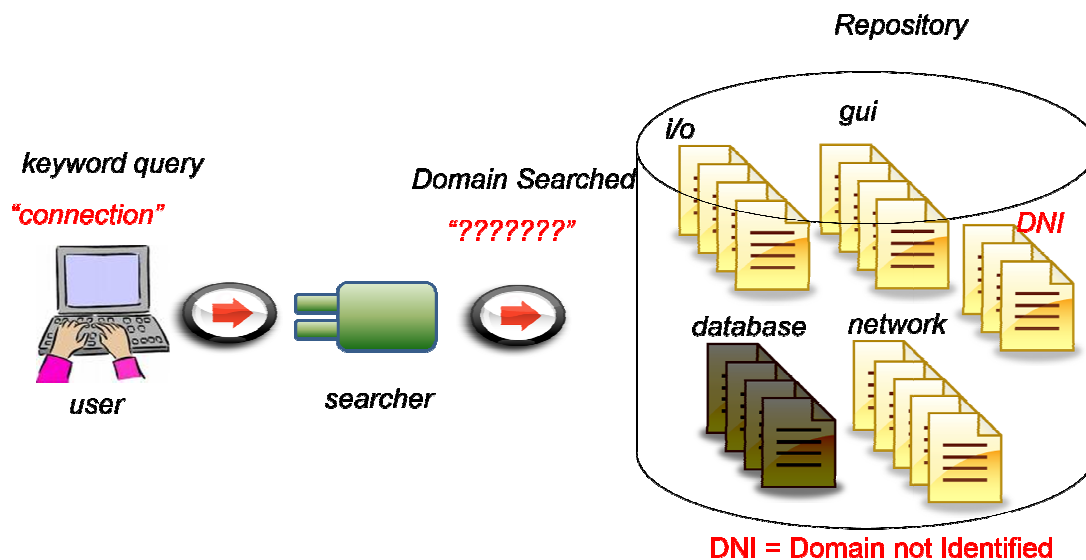


Figure 5.13. Query Reformulation

In the first scenario (Figure 5.13), the user searches for codes which contain the keyword "connection". At this point it is not possible to know the exact domain that the user desires to place his query, therefore, codes that perform network or database connection are expected to be returned in the search including other domains not expected yet. As soon as the search is performed, the codes which answer the query are returned as well as the related semantic

³⁶ [http:// www.eclipse.org/swt](http://www.eclipse.org/swt)

³⁷ <http://wiki.eclipse.org/JFace>

possibilities (domain terms) “*Database*” and “*Connection*”. At this moment, the user has extra information to drive his query in a specific domain that likely will bring codes closer to his need.

Internally the B.A.R.T server receives the query and redirects it to *searcher* and *reasoner* modules respectively. The *searcher* module retrieves all codes which contain the keyword “*connection*” while the *reasoner* module, utilizing the *Semantic Query Reasoner* component loads the infrastructure ontology and retrieves all technological domains that are associated with user query. Figure 5.14 shows a screenshot of the B.A.R.T Eclipse plug-in with the semantic possibilities.

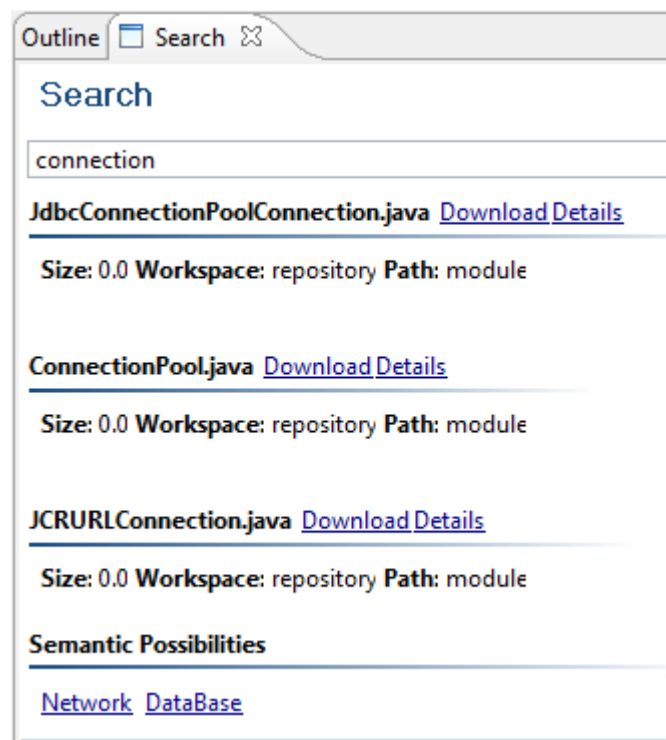


Figure 5.14. Screenshot of the B.A.R.T Eclipse plug-in

The present scenario exemplifies a polysemy case where the input keyword is associated with more than one domain. This problem may be characterized by ambiguous results among the codes returned that may lead the user to perform a “second” and “manual” search throughout the return collection selecting only those codes that belongs to the expected domain. With the semantic assistance, the possible domains are automatically showed helping

his to focus the search. Then by choosing a specific domain, the polysemy is overcome and the search is focused on the desired domain.

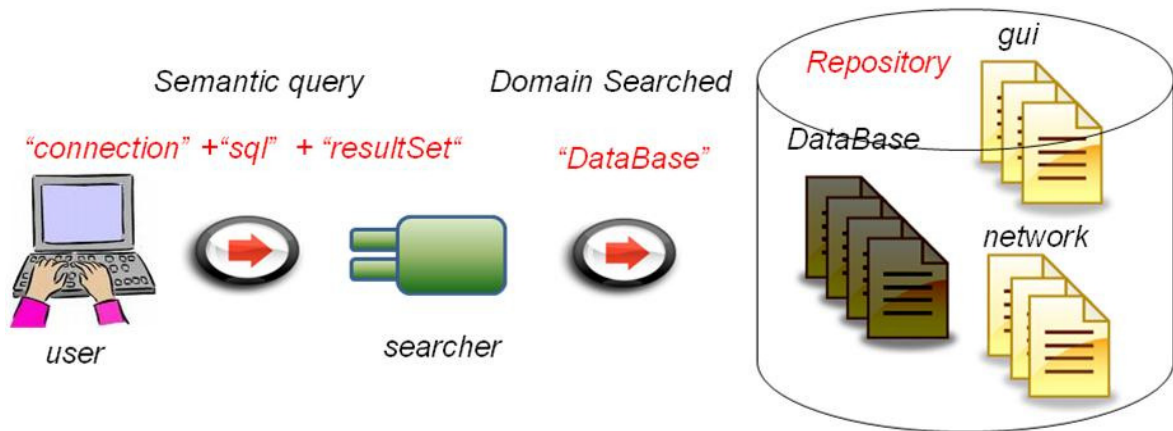


Figure 5.15. Query Expansion

In the second scenario (Figure 5.15), the "Database" domain is chosen, the current keyword "connection" is expanded with infra terms associated with the database domain, such as "sql", "resultSet". Besides that, the selected domain is kept for filtering the retrieval by those codes whose classification matches with the domain chosen. Internally, the B.A.R.T server receives the expanded query and redirects it to the *searcher* module which retrieves the codes which match with the expanded query and answer the semantic constraint.

It is important to state that the expanded query is a Boolean query e.g. "sql OR insert OR resultSet" that does not reduce the search performance because it carries out only over a fraction of the repository. This fraction is represented by codes which belong to the domain chosen and the D.N.I codes (those codes that do not have any classification associated). On the other hand, the retrieved codes which are classified receive a boost (value 2) on its score in order to augment its relevance over the D.N.I ones (Figure 5.16). The value 2 is anchored in the fact that the D.N.I files did not reached the minimal mark for being classified in any domain analyzed and hardly will answer the user need. The boost is a number which multiplies the original score achieved by index engine during the regular indexing process. As a result, the items which suffer the boost tend to appear firstly in list of retrieved files presented to user once they are ranked higher. Nevertheless, the value chosen for boosting the original

score must be chosen in a rationale according to the business logic of the program. Arbitrary boost definition may result in incoherent retrieval and unsuccessful search outcome. Figure 5.16 shows the retrieval process and boost value of the proposed solution.

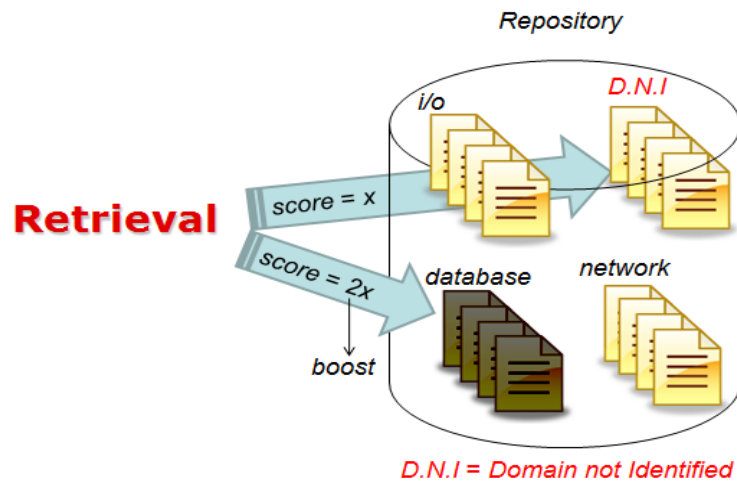


Figure 5.16. Retrieval of Semantic Search

Finally, after resolving the “semantic” retrieval the results are showed in the B.A.R.T Eclipse Plug-in ranked according to the relevance score. At this point, the probability of finding codes which deal with the functionality required is increased. Figure 5.17 shows the B.A.R.T Eclipse Plug-in with the codes which belongs to the “*Database*” domain.

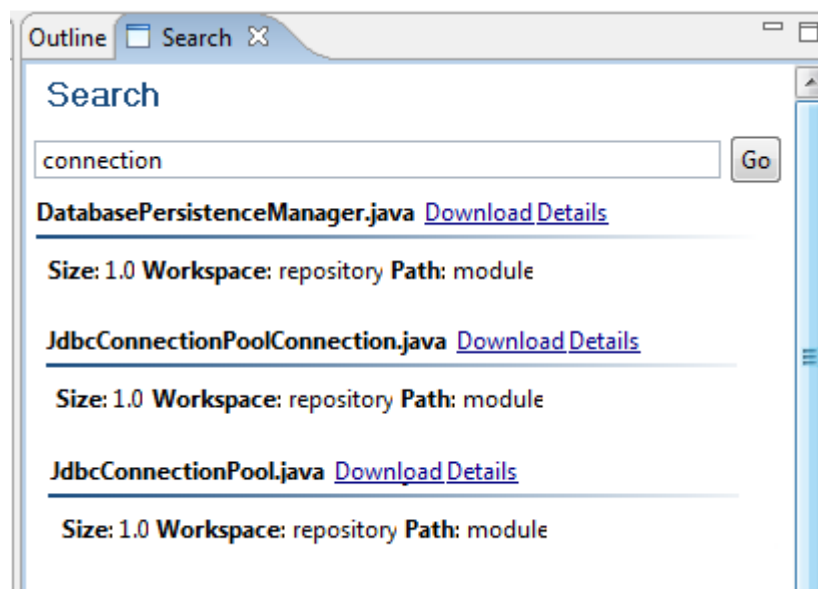


Figure 5.17. Final Search Results at B.A.R.T Search Plug-in

This Section presented a real scenario where the benefit acquired with the semantic assistance is evidenced; in the next Section the summary of this Chapter is presented.

5.7. Chapter Summary

This Chapter presented the main aspects of the Semantic Layer applied to the B.A.R.T search engine. The requirements, architecture and the set of technologies employed during its construction were discussed. Moreover, two new components which integrate the semantic layer were depicted: *Semantic Code Analyzer* and *Semantic Query Reasoner* and finally a simulation of the semantic search were detailed. The next Chapter presents the experiment performed to evaluate the semantic search engine with a set of real Java programs.



Semantic Search Engine Evaluation

Once the semantic layer was described and its initial implementation detailed, a experiment was performed in order to evaluate the benefits of the proposed solution. In this sense, some open source projects have been selected from remote repositories on the Internet in order to evaluate if this proposal is suitable for practical usage. This Chapter is organized as follows: Section 6.1 presents the methodology of the experiment, the questions that must be answered by its results and the variables to be analyzed. The projects used during the experiments are described in Section 6.2 as well as the formal hypotheses to be analyzed. Section 6.3 presents and analyzes the results of the experiments in terms of recall, precision and f-measure. Finally, Section 6.4 draws some conclusions on the findings of the evaluation.

6.1. Methodology

As stated in Chapter 5, the main goal of the *semantic layer* is to improve the search precision of the B.A.R.T search engine and consequently augment the relevance of the search returns. This achievement raises the chances of reuse because likely the codes returned will answer the user need.

In order to properly validate the proposed solution, the experiments must then reflect as closely as possible the proper environment where the proposed solution will be utilized out of the domestic test environment created during the construction of the tool.

In the best scenario the experiments should be performed on a set of real projects under development in order to raise appropriated feedbacks close from the reality. On the other hand, empirical evaluation would be unfeasible due the fact that such projects usually span across months respecting deadline and productivity marks. The inclusion of a trial tool in a real project would inhibit development teams since it would be seen as risk for the project schedule. Based on these reasons, an alternative and suitable scenario was set up with *source codes* from *remote repositories* available on the Internet able to be repeated by everybody else.

The methodology utilized for the evaluation, therefore, was to compare the previous search mechanisms (keyword and facet-based) of B.A.R.T search engine against the semantic one in terms of *precision* and *recall*. The folksonomy mechanism was not regarded due the fact that it depend on massive interaction of users to tag the classes in the repository beyond other variables particular to folksonomy such as the *tag relevance*.

To compare this semantic proposal with other related source code search engines presented in Chapter 5 might be another possible methodology; however, in general, the search engines analyzed did not let their test environment available or even have mentioned about. In spite of not performing the comparison, this methodology will not be forgotten and will be rethought for future works.

The following Subsection details the traditional information retrieval approach used for evaluating the semantic layer in the B.A.R.T search engine.

6.1.1. Information Retrieval Evaluation

The usual approach for evaluating the performance of information retrieval systems is based in terms of the *precision* and *recall* metrics by utilizing a large dataset along with a set of queries and expected responses [Baeza-Yates and Ribeiro-Neto, 1999].

According to this approach, to know the expected source code results is indispensable for calculating desired metrics, therefore, instead of randomly picking up codes from unknown projects, *known projects* with good documentation that help to identify and separate the relevant codes from the

ordinary ones are quite advisable. In addition, being familiar with the expected source codes it is easier to create appropriate *queries* and faster to realize possible faults.

To compare the overall results, this evaluation regards values close to 50% for recall and 20% for precision as satisfactory rates because are referenced from other authors [Frakes and Pole, 1994] [Ye and Fischer, 2002]. On the other hand, the achieved values will not be considered in the formal hypotheses of the experiments since they are exclusively values for reference. The formal hypotheses will be detailed in the following Subsection.

6.1.2. Evaluation of the Semantic Layer

This Section describes in details the goal, the variables, the experiment environment, the hypotheses and the results achieved in the evaluation of the semantic layer of the B.A.R.T search engine. It is important to stress that the methodology adopted, is strongly inspired by existing methodologies for information retrieval [Baeza-Yates and Ribeiro-Neto, 1999].

6.1.3. Goal

The goal of this section is to evaluate the semantic search engine from the point of view of the proposal uncertainties. In order to overcome the doubts about the proposed solution, some specific questions were raised:

1. “Does the semantic layer have contributed for increasing the recall of B.A.R.T search engine?”
2. “Does the semantic layer have contributed for increasing the precision of B.A.R.T search engine?”
3. “Is the semantic proposal a viable and practical mechanism to be part of the B.A.R.T search engine?”

After the test execution, it is expected to have all of the questions properly answered; however, independent of the findings, they will not be regarded as **conclusive** considering that more experiments must be performed

to proof its validity. The achieved responses will in fact **indicate** if the efforts of the RiSE group in search mechanisms are in the right direction.

6.1.4. Variables

Experiments usually employ a set of variables that link *causes* and *effects* [Perry et al., 2000], for this experiment *independent* and *dependent* variables were taken into account:

- *Independent variables* are attributes actively manipulated when comparing different situations.
- *Dependent variables* are the outputs whose values are expected to change according to changes to the independent variables.

In the context of this experiment, the independent variables refer to the search mechanisms evaluated in the experiment such as: *keyword-based*, *facet-based*, the combination between keyword and facet, *and semantic*. The results from previous search mechanism (*keyword and facet-based*) will be compared against the semantic one to compare the evolution of them. Thus, the independent variables are: *keyword*, *facet-based*, *keyword + facet-based*, *semantic*.

The dependent variables are the *precision*, *recall* and *f-measure* that correspond to the harmonic mean of *precision* and *recall*. In the following, each metric is detailed as well as the formula for its calculus:

- **Precision** is the fraction of a search output that is relevant for a particular query. Its calculation, hence, requires knowledge of the relevant and non-relevant hits in the evaluated set of documents [Clarke and Willett, 1997]. Thus it is possible to calculate absolute precision of search engines which provide an indication of the relevance of the system. In the context of the present study precision is defined as:

$$\text{Precision} = \frac{\text{Sum of the scores of codes retrieved by a search engine}}{\text{Total number of results evaluated}}$$

- **Recall** is the ability of a retrieval system to obtain all or most of the relevant documents in the collection. Thus it requires knowledge not just of the relevant and retrieved but also those not retrieved [Clarke and Willett, 1997]. The recall value is thus defined as:

$$\text{Recall} = \frac{\text{Total number of codes retrieved by a search engine}}{\text{Total number of results evaluated}}$$

- **F-measure** is the weighted harmonic mean of precision and recall. As the alpha value increases, the weight of recall increases in the measure. The formula of F-measure is as follows:

$$\text{F-Measure } \alpha = \frac{(1 + \alpha) * \text{precision} * \text{recall}}{((\alpha * \text{precision}) + \text{recall})}$$

To this experiment, the *precision* and *recall* rates have the same factor of importance, then, the closer a search mechanism is of 1.0, better it is. However, this only happens if both *precision* and *recall* are high. In case of high disparity between *precision* and *recall* rates, *f-measure* tends to zero, indicating that this mechanism does not play one of these criteria very well.

6.2. Experiment Configuration and Instantiation

Previous search and retrieval experiments of the RiSE group such as [Garcia et al., 2006b] and [Vanderlei et al., 2007] have definitely collaborated to this one. In addition to the important points such as structure and the required issues to be analyzed, the other points to be improved were also observed and tried to be overcome in this instantiation. Both experiments shared a common dataset (or test database) what, in fact, this is positive because they can show the evaluation of the search mechanisms with more confidence; on the other hand, over than **1000** Java files were utilized without total knowledge of the dataset. By this

reason, for this experiment the number of classes in the dataset was reduced to **100** with the condition that each class should be properly analyzed, classified and catalogued in standard dataset for future reuse.

In terms of *precision* and *recall*, the amount of class does not represent valuable information because this metrics are concerned in revealing if the search engine returns the relevant items or not. Therefore, for big or small datasets, the expected items have necessarily to be known and this process logically will be more expensive for the big ones [Clarke and Willett, 1997]. Thus, for this experiment the number of classes was drastically reduced aiming to recognize minimally each class and then to perform a more accurate evaluation.

In this sense, the scenario used for the experiment was formed with 100 Java classes from 10 distinct open-source projects on the Internet (Appendix presents the projects utilized). The project choices are justified by the fact they cover a broad range of domains and all of them are written in the Java programming language besides being open-source which allows repetition of the experiment. The whole classes were carefully analyzed in order to extract its *domain*, *keywords* and *facets* to be utilized as filter for the search mechanism evaluated in the experiment (*independent variables*). The entire commitment since the dataset creation until the query definition was tackled by 6 RiSE members in the period between October, 1st and October, 30th 2007 with common agreement about the information raised. The objective of this approach was to avoid the addicted test environment used during the software development.

The information extracted from the dataset analyze as well as the expected results were placed in a spreadsheet to document the entire statistics of the evaluation. Table 6.1 shows part of the information extracted from the source code analysis.

Table 6.1. Source Code Information

Class Name	Host Project	Repository	Domain	Technology	Keyword
Painter.java	SoapUI	SourceForge.net	GUI	J2SE	paint area
XmlUtils.java	SoapUI	SourceForge.net	XML	J2EE	xml parse

During the tests, the *domain* information is consumed by the semantic mechanism, *technology facet* is used as filter for the facet-based mechanism and the *keywords* for the traditional keyword search mechanism. The following step after having the entire dataset documented was to determine the queries and their respective expected results. The appendix of this dissertation outline the queries utilized during the test execution.

6.2.1. Experiment Hypotheses

In order to formalize the experiments some hypotheses were raised. The *null hypotheses (Ho)*, i.e. the hypotheses intended to be rejected will indicate if the semantic approach or its combination with other search mechanisms do *not* have contributed for increasing the precision and recall of the B.A.R.T search engine.

Recall Hypotheses

- **Hoa:** the keyword-based mechanism has higher recall than the semantic;
- **Hob:** the facet-based mechanism has higher recall than the semantic;
- **Hoc:** the facet-based + keyword mechanisms have higher recall than the semantic.

Precision Hypotheses

- **Hod:** the keyword-based mechanism has higher precision than the semantic;
- **Hoe:** the facet-based mechanism has higher precision than the semantic;
- **Hof:** the facet-based + keyword mechanisms have higher precision than the semantic;

F-measure Hypotheses

- **Hog**: the keyword-based mechanism has higher f-measure than the semantic;
- **Hoh**: the facet-based mechanism has higher f-measure than the semantic;
- **Hoi**: the facet-based + keyword mechanisms have higher f-measure than the semantic;

By rejecting these hypotheses, the following *alternative hypotheses (H1)* are favored:

Recall Hypotheses

- **H1a**: the semantic mechanism has higher recall than the keyword-based;
- **H1b**: the semantic mechanism has higher recall than the facet-based;
- **H1c**: the semantic mechanism has higher recall than the facet-based + keyword;

Precision Hypotheses

- **H1d**: the semantic mechanism has higher precision than the keyword-based;
- **H1e**: the semantic mechanism has higher precision than the facet-based;
- **H1f**: the semantic mechanism has higher precision than the facet-based + keyword;

F-measure Hypotheses

- **H1g**: the semantic mechanism has higher f-measure than the keyword-based;
- **H1h**: the semantic mechanism has higher f-measure than the facet-based;
- **H1i**: the semantic mechanism has higher f-measure than the facet-based + keyword;

In the optimistic view, **H1g**, **H1h** and **H1i** hypotheses are expected to be favored, besides the entire *null hypotheses* being rejected. However it is known that the semantic mechanism tends to decrease the recall since the search focuses on classes with belong to a specific domain instead of the entire repository. Therefore a bad classification could prejudice the recall and consequently put the f-measure rate down. In the following Section, the results achieved in the experiments are described.

6.3. The Evaluation Results

This Section presents and discusses the results of the experiment achieved with the configurations of the search mechanisms described in the Section 5.1.1. The **Quantitative Evaluation** was divided in three independent analyses: *recall*, *precision* and *f-measure*. For each analysis, it was calculated the *arithmetic average*, the *standard deviance* and the *variance* for the total of queries performed.

- **Standard Deviance:** the semantic mechanism has higher f-measure than the keyword-based;
- **Variance:** the semantic mechanism has higher f-measure than the facet-based.

6.3.1. Recall Results

Table 6.2 shows the *independent variables* against the arithmetic average of recall, standard deviance and variance.

Table 6.2. Achieved Recall Rates

Search Mechanism	Recall	Standard Deviance	Variance
keyword	0,81	0,21	0,04
facet-based	0,49	0,33	0,09
keyword + facet-based	0,41	0,29	0,06
<i>semantic</i>	0,76	0,15	0,02

According to Table 6.2, the keyword mechanism has achieved the highest recall followed by the semantic, facet-based and the facet-based + keyword mechanism. The recall of the keyword and semantic mechanisms was higher than the reference value obtained by other authors (50% for the recall) [Ye and Fischer, 2002] [Frakes and Pole, 1994] which are regarded as satisfactory. On the other hand, the facet-based and its combination with the keyword mechanism need to improve their recall since their values are below the satisfactory average. In general, the whole search mechanisms had 35% of standard deviance and very low variance which means that their findings were close to the recall average and not disparity was evidenced. Although the recall of the semantic mechanism is worse than the keyword one, the difference between both is low which indicates that semantic mechanism even with the restriction of the domain did not left many source codes behind. From this perspective, high precision is expected since the search is focused on a specific set of classes.

According to the results obtained and formal hypotheses which analyze the recall, both *null hypotheses* **H_{0b}** and **H_{0c}** are rejected and as a consequence the *alternative hypotheses* **H_{1b}** and **H_{1c}** are privileged.

6.3.2. Precision Results

Table 6.3 shows the *independent variables* against the *arithmetic average* of the precision, the standard deviance and the variance.

Table 6.3. Achieved Precision Rates

Search Mechanism	Precision	Standard Deviance	Variance
keyword	0,57	0,17	0,044
facet-based	0,36	0,27	0,11
keyword + facet-based	0,76	0,22	0,08
<i>semantic</i>	0,79	0,14	0,02

According to Table 6.3, the semantic mechanism achieved the highest precision followed by the facet-based + keyword, keyword and facet-based mechanism. Hopefully the whole mechanisms have overcome the reference value obtained by other authors (20% for precision), even although the facet-based value (36%) was very close to this. Nevertheless when comparing the facet-based mechanism with the other ones, it is realized that improvements are urgently needed. Either the facets available or those used in the evaluation might not be quite representative for the dataset utilized. Once again, the whole search mechanisms had low standard deviance (no more than 27%) and very low variance which means that the findings were close to the precision average and not disparity among the obtained results was evidenced.

The belief of high precision by the semantic mechanism is happily evidenced supported by the lowest standard deviance and variance jointly. Though this, it is seem when the search is focused on the specific domain the results in fact are closer to user necessity, however, to achieve such optimistic result an efficiency classification must have been performed.

According to the results obtained and formal hypotheses which analyze the precision, the *null hypotheses* **Hod**, **Hoe** and **Hof** are rejected and as a consequence the *alternative hypotheses* **H1d**, **H2e** and **H3f** are privileged.

6.3.3. F-Measure Results

Table 6.4 shows the *independent variables* (search mechanisms) against the *arithmetic average* of the f-measure, the *standard deviance* and the *variance*.

Table 6.4. Achieved F-Measure Rates

Search Mechanism	F-Measure	Standard Deviance	Variance
keyword	0,65	0,15	0,02
facet-based	0,28	0,16	0,02
keyword + facet-based	0,45	0,19	0,03
<i>semantic</i>	0,75	0,09	0,01

According to Table 6.4, the semantic mechanism achieved the highest f-measure rate followed by the keyword, facet-based + keyword and facet-based mechanism. The main reason for this may be regarded by the balance in recall and precision findings of the semantic mechanism. In addition to the pleasing values achieved for the recall and precision, the low discrepancy between them has contributed for the highest f-measure rate. In opposite, the same cannot be said to the keyword + facet-based which had low recall (41%) and high precision (76%); certainly such difference pulls the f-measure down.

Although no reference value is used to compare with the obtained values, both keyword and semantic mechanism has achieved values above 50% of the maximum expected and then may be considered as satisfactory. Again, for the whole mechanisms, the low standard deviance and variance was observed which means that no disparity between the f-measure findings and its average has happen.

According to the results obtained and formal hypotheses which analyze the f-measure, the *null hypotheses* **H_{0g}**, **H_{0h}** and **H_{0i}** are rejected and as a consequence the *alternative hypotheses* **H_{1g}**, **H_{1h}** and **H_{1i}** are privileged.

6.4. Final Considerations and Discussion

Among the hypotheses analyzed, the **H_{0a}** which states that the keyword-based mechanism has higher recall than the semantic was the unique *null hypothesis* privileged. On the other hand, the whole *alterative hypotheses* were favored, except for the **H_{1a}** which states that the semantic mechanism has higher recall keyword-based mechanism. Although the recall of the semantic mechanism had been inferior to keyword mechanism, the disparity between them was minimally what does not guarantee the *null hypothesis* **H_{0a}**.

Table 6.5 summarizes the mean of recall, precision and f-measure achieved during the experiment evaluation. If looking at only the precision value, it is pleasing to see the semantic approach has reached the highest precision rate among the mechanisms compared. On the other hand the recall of the semantic mechanism was below the keyword which means that adjustment in the components of the semantic layer must be done.

Table 6.5. Achieved Metrics

Search Mechanism	Recall	Precision	F-Measure
keyword	0,81	0,57	0,65
facet-based	0,49	0,36	0,28
keyword + Facet-based	0,41	0,76	0,45
<i>semantic</i>	0,76	0,79	0,75

In a general overview, the recall and precision of the semantic approach rates have reached reference values obtained by other authors (50% recall and 20% for precision) [Frakes and Pole, 1994] [Ye and Fischer, 2002]. This indicates that the semantic layer place the B.A.R.T search engine in the same level of others search engines allowing be comparable in terms of recall and precision. Although the comparison with other search engines can give a real feedback about the B.A.R.T search engine effectiveness in the trade market or in the research community, first the ongoing mechanisms like this (semantic) must be more explored until obtain the status of “releasable”.

Moreover, in spite of achieving optimistic results, it is not safe to affirm which mechanism is the best or that the semantic mechanism excludes the usage of other mechanism which had less performance in the experiment. Indeed, further experiments with other dataset (with different projects, domains and size) and other queries are needed in order to provide more concrete information about the tool evolution. Only the average of continuous evaluations could then express the true effectiveness of the tool. On the other hand, the results so far indicate the research is on the right track.

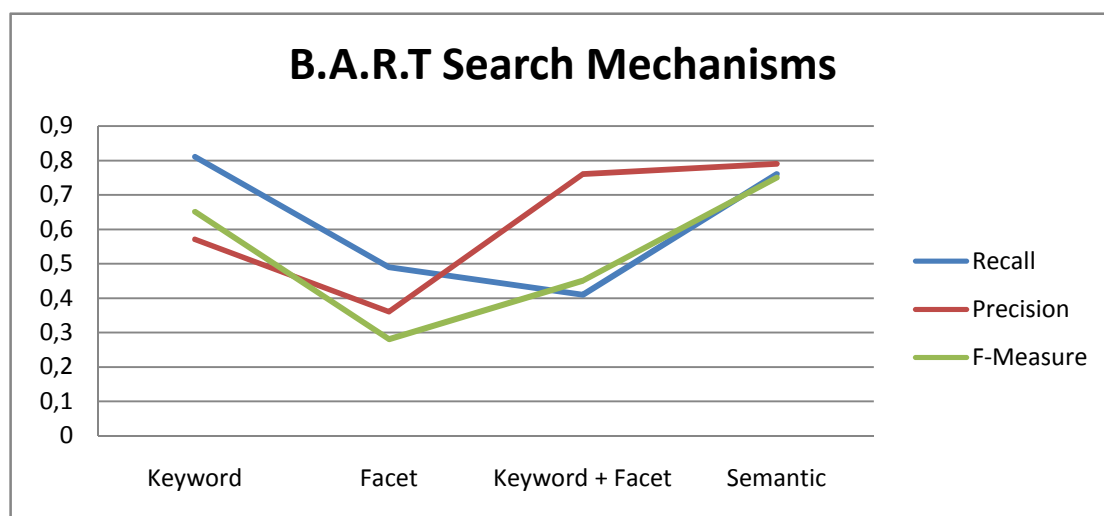
**Figure 6.1. The evolution of B.A.R.T search mechanisms**

Figure 6.1 shows line graphic that outlines the evolution of search mechanisms of B.A.R.T search engine in terms of recall, precision and f-measure. The B.A.R.T evolution has been evidenced since the search mechanisms have been incorporated, except for the facet-based mechanism (and its combination) that played the initial evolution down. This motivated the RiSE group to allocate some members only to research about the facet-based approach. Alike the keyword mechanism, the semantic approach has contributed to converge recall and precision rates which elevates consequently the f-measure.

6.4.1. Problems Found

By analyzing the final test results, the semantic approach was ranked higher than the keyword and facet-based mechanisms; however, the evaluation has revealed some problems:

- **Low recall** – The recall of semantic mechanism was lower than the keyword mechanism which favored the *null hypothesis* **H₀**: “*the keyword-based mechanism has higher recall than the semantic*”. Two causes might have contributed for this: bad classification accuracy and poor query expansion. In the first case, some classes not or wrongly classified might have been ignored during the search; for the second cause, the expanded query might have been constructed with inappropriate terms so that classes were not retrieved. Although the “*High precision and recall*” requirement (Chapter 6.1) had been fulfilled partially, special attention must be given to the knowledge base construction and the ontology vocabulary.
- **Amount of terms that composes the expanded query** – As described in Chapter 5, once the user chooses a domain, the initial query is expanded with other related terms. However, to achieve the best retrieval with better recall, not necessarily whole terms of a given domain must be elected to compose the expanded query. In this sense, some rationale must be followed taking into account the user context, the chosen domain and previous queries.

- **D.N.I (Domain not Identified) size** – The fraction between D.N.I size and remain classes of the repository classes must be considered in the situation when the D.N.I size is higher. In this scenario, the semantic search depends massively on the query expansion that correspond to a Boolean query e.g. “*term 1 OR term 2 OR term 3*” and if applied to the largest part of the repository may considerably affect the search performance. One possible solution for this problem could be to change the Boolean query to other arrangements that could preserve search performance. On the other hand, when the D.N.I size is lower than the remain classes of the repository; the semantic search tends to work properly.

The next Section answers the questions posed in Section 6.4.1.

6.4.2. Goal Questions

Section 6.1.1 posed some questions about possible benefits achieved with semantic layer in the B.A.R.T search engine.

- The first question asked: “*Does the semantic layer have contributed for increasing the recall of B.A.R.T search engine?*” According to the obtained results the semantic search did not presented expressive contribution for increasing the recall, indeed, a slight decrease was observed in spite of the results achieved were considered satisfactory.
- The second question asked: “*Does the semantic layer have contributed for increasing the precision of B.A.R.T search engine?*” According to the obtained results the semantic search presented improvements in the precision. In fact, the semantic layer achieved its goal by retrieving relevant source codes closer to user need.
- The third question asks: “*Is the semantic proposal a viable and practical mechanism to be part of the B.A.R.T search engine?*” According to the general evaluation of the proposed solution, the semantic proposal may be regarded as a viable and practical mechanism to make part of the B.A.R.T search engine. Although improvements are required, the semantic mechanism configures

another mean to assist the developers to find codes more relevant and consequently increases the chances of reuse.

Next Section summarizes this Chapter presenting all aspects involved in the evaluation process.

6.5. Chapter Summary

This Chapter presented the experiments conducted to evaluate the semantic version of the B.A.R.T search engine in terms of the precision and recall on a set of real projects. The results showed that although some tuning for the information retrieval engine is still needed, the proposed approach consists on a valuable resource for increasing the precision of B.A.R.T search engine and consequently in achieving a higher reuse activity level.

Next Chapter concludes this dissertation by summarizing the analysis performed on this Chapter, reviewing some related works, pointing directions for future enhancements to the environment and presenting some final considerations.



Conclusion

Once the semantic layer has been defined, implemented and evaluated based on the research of software reuse, information retrieval and semantic web areas, some conclusions and comparisons can be drawn and directions to future work pointed out.

This Chapter is organized as follows: Section 7.1 summarizes the achieved goals of the work and Section 7.2 presents a comparison with some related works. Section 7.3 points out some directions for future works unexplored by this work and, finally, Section 7.4 contains a closing discussion on the topics covered in this dissertation.

7.1. Achieved Goals

The main contribution of this dissertation is the proposed a semantic layer applied to a keyword-based search engine in order to increase the precision of search returns. The proposed solution utilizes a domain ontology for enhancing the construction of the query with related terms and a machine learning technique for source code classification. The implemented proposal configures a **viable** and **practical** solution for being utilized in an industrial scenario in favor of source code reuse. Through the real experiment, it was evidenced the increase of precision during the code searches and this finding goes against the semantic conceptual gap between user needs and machine understanding.

In addition, other specific findings after getting the work concluded are:

- Integration among different areas such as Software Reuse, Information Retrieval and Semantic for developing comprehensive solution intended for practical usage in software development factories;

- Use of semantic technologies such as ontologies to overcome the process of asset location and semantic conceptual gap. The ontology vocabulary was used to expand the user query and then increase the precision of the search;
- Extension of a source code information retrieval tool (B.A.R.T search engine) with a semantic layer built under component-based development in favor of reuse of new development. The proposed semantic layer primes for two main non-functional requirements: **reusability**, so that the proposed *semantic components* built under the *Component-Based Development* can be reused by other applications and **extensibility** so that the architecture proposed for the semantic components can be easily expanded with other functional modules;
- The proposed semantic layer of the B.A.R.T search engine was built based on the reuse of existing technologies, with the goal of closing the gaps identified during the research phase;
- The evaluation results show that although some fine tuning is still needed, the semantic version of the B.A.R.T search engine have increased the precision of the tool. This feature indicates that the B.A.R.T search engine under semantic assistance increases the chance of code reuse since the developers will retrieve source codes closer to their needs; and
- Evaluation of the proposed solution based on a formal experiment that can be repeated and extended with other dataset in terms of content and size.

7.2. Related Work and Research Foundations

The products of this work are a result of a careful research on three interrelated fields: *Software Reuse*, *Information Retrieval* and *Semantic Web*. The semantic version of B.A.R.T search engine have concentrated its efforts on these three pillars to provide an integrated set of functionalities aiming to increase the reuse activity in organizations. This Section discusses some related works and concerning on these three fields that have in some way inspired the definition of the proposed solution.

7.2.1. Software Reuse

Since the first breathing of *software reuse* [McIlroy, 1968], this field has gained attention among the research community as well as in the industry. The rationale on how producing software from existing pieces of codes previously built sounds like favorable financial savings for CEOs. The dissemination of reuse initiatives have encouraged software developers to reuse existing software assets instead of developing new ones, consequently, it results in less time needed to develop a system and then speed up the time-to-market.

Motivated by these premises, a process for the reuse startup have being proposed based on three main directions such as *Component-Based Development Process* [Heineman and Council, 2001] [Neto et al., 2004], *Domain Engineering* [Frakes et al., 1998, Almeida, 2007] and *Software Product Lines* [Atkinson et al., 2000]. In addition, a set of tools in promoting for facilitating the creation and accessing to the reusable software assets also have being developed such as *Component Search Engines* [Garcia et al., 2006b] [Ye and Fischer, 2002], *Reuse Repositories* [Burégio, 2006], *Reengineering Tools* [Brito, 2007], *Domain Analysis Tools* [Lisboa et al., 2007].

All of these contributions appears jointly with systematic politics, reuse best practices and certification process [Alvaro et al., 2006] in order fit with traditional process of software development.

7.2.2. Information Retrieval

Because of the synergy of software reuse and information retrieval, the last has been the focus of many works involving software reuse [Henninger, 1994] [Thomason et al., 2000] [Ye and Fischer, 2002] [Sugumaran and Storey, 2003]. In spite of not showing all of them, a set of the most influent is described in the following.

The *CodeFinder* proposed by Henninger (Henninger 1994) is a code searcher that uses query-construction methods for assisting users to define their needs when they do not know the exact terminology. The main concern behind the Henninger's tool is that locating software items is difficult, even for

well-informed software designers, when searching in large, complex, and continuously growing libraries. Similarly to the CodeFinder, the semantic version of B.A.R.T search engine tries to help users to narrow its search on the specific context through semantic suggestions about the domains which better fit with query input.

In 2000, (Thomason, Brereton et al. 2000) proposed the *CLARiFi*, a component-based system that provides an classification schema that identifies component properties important in the selection for a given task. Classification schemes implemented by automated categorization techniques are practical instruments for augmenting the knowledge about what is retrieved. Equally to Thomason et al., this dissertation has utilized an automatic text categorization employed to identify the technological domain handled by the source code. Such information is useful to identify cluster of domains and focus the search on specific areas of large reuse repositories.

Other very influent code search engine was the *CodeBroker* (Ye and Fischer 2002): a context-based retrieval tool based on user environment where the components returned took into account class syntax structures while user coding activity. The main concern behind the *CodeBroker* is the distance between user need and the computational understanding of the query or “*Semantic Conceptual Gap*”. Following this premise, this semantic proposal has utilized a domain ontology to properly reformulate the original user queries in other that may better represent the user need.

In 2003, Sugumaran and Storey (Sugumaran and Storey 2003) presented *A Semantic-Based Approach to Component Retrieval* to meet user’s requirement taking into account a domain ontology which included domain terms, definitions, and relationships besides other domain-specific terms. Sugumaran and Storey have utilized ontology semantics for improving the precision of its source code search engine. They have modeled the *auction* domain ontology that was utilized to find methods and attributes of classes that matched with the ontology vocabulary. The use of a domain ontology converges Sugumaran and Storey’s proposal to this one at the point that it tries to eliminate the semantic conceptual gap between the subjective user queries and the machine understanding about the request. Although the ontologies of Sugumaran's work and this proposal were modeled for different domains, the

purpose was the same: to improve the source search engines with the use of domain ontologies. The use of ontologies has been significantly possible thanks to the advances of the Semantic Web field discussed in the following.

7.2.3. Semantic Web

The semantic web was designed as an information space, with the goal that it should be useful not only for human communication, but also for machine participation and help [Berners-Lee et al., 2001]. This belief has motivated the development of *Markup Ontology Web Languages*: a formal data model which supports the semantics of the business entities and allows software applications to process and manipulate its content [Bruijn, 2003]. Although not only restrict to web development, the *Markup Ontology Web Languages* were employed to build different-purpose applications such as the semantic search engines.

In this context, outstanding related works have inspired this dissertation such as OntoSeek [Guarino et al., 1999], a system designed for content-based information retrieval from online yellow pages and product catalogs. Guarino et al. believe that representations of structured content coupled with linguistic ontologies can increase both recall and precision of search engines. Guarino's approach is concerned with solving semantic-match problems by using linguistic ontologies such as WordNet³⁸ and structured representation formalisms. Similarly to the approach proposed, OntoSeek provides means of semantic matching between queries and resource descriptions by making use of ontology.

Like OntoSeek, the *Semantic Web Search Engine* (SWSE) uses ontologies to provide categories around the input query in order to contextualize the search. Parallel to this semantic proposal, SWSE also employs ontology reasoning in order to enhance the information retrieval capabilities. Another very important related work is the *Semantic Search Engine for the Storage Resource Broker* [Jeffrey and Hunter, 2006], a semantic version of an keyword-based Storage Resource Broker (SRB) system utilized for retrieving distributed data collections of software assets such as source code, documents, models, etc.

³⁸ <http://wordnet.princeton.edu>

In its original version, the keyword-based SRB search method had some limitations because the users had to have a precise knowledge of the metadata schema and vocabularies used by a particular scientific community. This problem lead the company to enhance the system's search capabilities by implementing a semantic layer built on top of an OWL ontology, RDF instance data and a Jena reasoning engine to enable easier and more sophisticated searching of heterogeneous data stored using SRB. In the same sense, this proposal also has utilized OWL ontology for augmenting the search capabilities of the keyword-based mechanism.

The next Section outlines future works proposed for the current development.

7.3. Future Work

In spite of the intense commitment to develop the semantic version of the B.A.R.T search engine, some enhancements are visualized, since the initial goal was to demonstrate the viability of this approach in an academic level. In this fashion, some important aspects that were left out of this version are enumerated:

- **Knowledge Base Feeding** – To provide an user interface that enables domain specialists to extend and maintain the *knowledge base* with others technical domains handled by source code. This feature will allow quick refinement and population of the *knowledge base* whenever improvements are necessary for the search;
- **Semantic and Folksonomy Matching** - A very promising feature is that the Folksonomy mechanism could be applied to populate the ontology vocabulary in a controlled mode where the suggested tags could be validated by a domain specialist and then inserted in the current ontology model. Through this functionality, the user tags would be placed in a specific domain following the taxonomic hierarchy respecting the semantic constraints. At that point, the tool already provides an interface for tagging the (provided) semantic possibilities; however, no assessment in done;

- **Artifact Types Support** - New artifact types and formats must be supported in order to provide the search with a broader coverage range. Documents written in natural language, in particular, certainly will necessitate the extension of the ontology structure considering that new ontology properties will be required;
- **Data Mining for Rules Suggestion** – The most powerful *Web Ontology Mark-up Languages* such as RDF, OWL and DAML-OIL execute inference over predefined rules written in text files. For performing the inference properly, the content of the rule must respect the ontology model and, in general, vary according to formal business constraints or empirical awareness. Data mining techniques, such as *Association Rules*, permit to extract patterns of relationships between entities within a specific context [Agrawal and Srikant, 1998]. The repetition of such patterns is necessary to the establishment of well-founded rules. Based on this, such rules under a specialist assessment may configure an opportunistic mechanism to the creation of new rules without human interference [Agrawal and Srikant, 1998];
- **Use of Other Text Categorization Technique** - In the current version, the semantic classification takes place through the *Nayve Probabilistic Method*, nevertheless, other mechanisms such as *Supported Vector Model* or *Latent Semantic Analysis* should be used instead. The objective is to continuously be enhancing the efficiency of the source code categorization and consequent search accuracy;
- **Semantic Granularity** - The semantic level managed by this dissertation refers to technological domains handled by source codes similar to the Java packages such as *math*, *network* and *security*. Although this domain information is useful for the improvement of the search, the next step can be done in the direction of *methods* and *attributes*. By increasing the granularity of the code analysis, the information about the code functionalities will be more precise, increasing, therefore, knowledge about what is required; and
- **New Experiments** - New experiment configurations for further evaluation may be tackled: repetition of the experiment with new dataset

through the use of new projects or comparing the semantic search engine against related tools by sharing a common experiment environment.

7.4. Concluding Remarks

The products of this work are a result of a careful research on three interrelated fields: software reuse, information retrieval and semantic web. The semantic layer was built on these three pillars to provide source code search more precise and consequently increase the chance of reuse by the user. In this Section, the findings of the studies in these three fields were presented, although no existing solution is integrally equivalent to the proposed solution, this Section also discussed some related works that in some way have inspired the development of the semantic layer and finally introduced the future works planned to be realized in the future versions.



ferences

- [Agrawal and Srikant, 1998] Agrawal, R. and Srikant, R. (1998). **Fast algorithms for mining association rules**. San Francisco, Morgan Kaufmann Publishers.
- [Almeida, 2007] Almeida, E. S. (2007), **RiDE: The RiSE Process for Domain Engineering**, Informatic Center, Federal University of Pernambuco (sandwich period at Universität Mannheim), Recife,
- [Almeida et al., 2004] Almeida, E. S., Alvaro, A., Lucrédio, D., Garcia, V. C. and Meira, S. R. L. (2004). **RiSE Project: Towards a Robust Framework for Software Reuse**. IEEE International Conference on Information Reuse and Integration (IRI), Las Vegas, USA, IEEE/CMS, p. 48-53.
- [Almeida et al., 2005a] Almeida, E. S., Alvaro, A., Lucrédio, D., Garcia, V. C. and Meira, S. R. L. (2005a). **A Survey on Software Reuse Processes**. IEEE International Conference on Information Reuse and Integration (IRI), Las Vegas, Nevada, USA, p. 66-71.
- [Almeida et al., 2005b] Almeida, E. S., Alvaro, A. and Meira, S. R. L. (2005b). **Key Developments in the Field of Software Reuse**. 15th PhDOOS Workshop, Glasgow, Scotland, p. 10-12.
- [Alvaro et al., 2006] Alvaro, A., Almeida, E. S. and Meira, S. R. L. (2006). **A Software Component Quality Model: A Preliminary Evaluation**. 32nd IEEE EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), Component-Based Software Engineering Track, Cavtat/Dubrovnik, Croatia, p. 28-35.
- [Atkinson et al., 2000] Atkinson, C., Bayer, J. and Muthig, D. (2000). **Component-Based Product Line Development: The Kobra Approach**. First Product Line Conference (SPLC), Kluwer International Series in Software Engineering and Computer Science, Denver, Colorado, USA, p. 19.
- [Atkinson et al., 1989] Atkinson, M., Bancilhon, F., DeWitt, D., Dittrich, K., Maier, D. and Zdonik, S. (1989). **The object-oriented database system manifesto**. 1st International Conference on Deductive and Object-Oriented Databases, Kyoto, Japan, Elsevier Science, p. 40-57.
- [Baeza-Yates and Ribeiro-Neto, 1999] Baeza-Yates, R. and Ribeiro-Neto, B. (1999). **Modern Information Retrieval**, ACM Press / Addison-Wesley.

-
- [Basili et al., 1996] Basili, V. R., Briand, L. C. and Melo, W. L. (1996). **How Reuse Influences Productivity in Object-Oriented Systems**. Communications of the ACM Vol.(39), No. 10, p. 104--116.
- [Belkin and Croft, 1992] Belkin, N. and Croft, B. (1992). **Information Filtering and Information Retrieval**. Communications of the ACM Vol.(35), No. 12, p. 29-37.
- [Berners-Lee, 1996] Berners-Lee, T. (1996). **WWW:Past, Present and Future**. IEEE Computer Vol.(29), No. 10, p. 69 - 77.
- [Berners-Lee et al., 2001] Berners-Lee, T., Hendler, J. and Lassila, O. (2001). **The Semantic Web**. Scientific American Vol.(284), No. 5, p. 34-43.
- [Breslin et al., 2005] Breslin, J. G., Harth, A., Bojars, U. and Decker, S. (2005). **Towards Semantically Interlinked Online Communities**. 2nd European Semantic Web Conference, Heraklion, Greece, Springer-Verlag, p. 500-514.
- [Brito, 2007] Brito, K. d. S. (2007), **LIFT - A Legacy InFormation retrieval Tool**, M.Sc. Dissertation, Federal University of Pernambuco, September, 2007.
- [Bruijn, 2003] Bruijn, J. d. (2003). **Using Ontologies - Enabling Knowledge Sharing and Reuse on the Semantic Web**. Austria, Digital Enterprise Research Institute (DERI).
- [Burégio, 2006] Burégio, V. A. A. (2006), **Specification, Design, and Implementation of a Reuse Repository**, Federal University of Pernambuco, August, 2006.
- [Calado and Ribeiro-Neto, 2003] Calado, P. P. and Ribeiro-Neto, B. (2003). **An Information Retrieval Approach for Approximate Queries**. IEEE Transactions on Knowledge and Data Engineering Vol.(15), No. 1, p. 237-240
- [Chen et al., 2007] Chen, L., Shadbolt, N. R. and Goble, C. A. (2007). **A Semantic Web-Based Approach to Knowledge Management for Grid Applications**. IEEE Transactions on Software Engineering Vol.(19), No. 2, p. 283-296.
- [Clarke and Cormack, 1995] Clarke, C. and Cormack, G. (1995). **Dynamic Inverted Indexes for a Distributed Full-Text Retrieval System**. MT-95-01, T. R.
- [Clarke and Willett, 1997] Clarke, S. and Willett, P. (1997). **Estimating the recall performance of search engines**. ASLIB Proceedings Vol.(49), No. 7, p. 184-189.
- [Clements and Northrop, 1996] Clements, P. and Northrop, L. (1996). **Software Architecture: An Executive Overview**, Carnegie Mellon University, Software Engineering Institute.
- [Clements and Northrop, 2002] Clements, P. and Northrop, L. (2002). **Software Product Lines: Practices and Patterns**, Addison-Wesley.
- [Connolly et al., 2001] Connolly, D., Harmelen, F. v., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F. and Stein, L. A. (2001). **DAML+OIL (March 2001) Reference Description**. W3C Note
- [Decker et al., 2000] Decker, S., Mitra, P. and Melnik, S. (2000). **Framework for the Semantic Web: An RDF Tutorial** IEEE Internet Computing Vol.(4), No. 6, p. 68-73.
- [Devanbu et al., 1991] Devanbu, P., Brachman, R. J., Selfridge, P. G. and Ballard, B. W. (1991). **LaSSIE: A Knowledge-Based Software**

- Information System.** Communications of the ACM Vol.(34), No. 5, p. 34-49.
- [Devedzić, 2001] Devedzić, V. (2001). **Knowledge Discovery and Data Mining in Databases.** Handbook of Software Engineering and Knowledge Engineering. Singapore: 615-637.
- [Devedzić, 2002] Devedzić, V. (2002). **Understanding ontological engineering.** Communications of the ACM Vol.(45), No. 4, p. 136 - 144
- [Ding et al., 2004] Ding, L., Finin, T., Joshi, A., Pan, R., Cost, R. S., Peng, Y., Reddivari, P., Doshi, V. and Sachs, J. (2004). **Swoogle: a search and metadata engine for the semantic web.** 13th ACM international conference on Information and knowledge management, Washington, D.C., USA ACM Press, p. 652-659.
- [Edgington et al., 2004] Edgington, T., Choi, B., Henson, K., Raghu, T. S. and Vinze, A. (2004). **Adopting ontology to facilitate knowledge sharing.** Communications of the ACM Vol.(47), No. 11, p. 85-90.
- [Ezran et al., 2002] Ezran, M., Morisio, M. and Tully, C. (2002). **Practical Software Reuse,** Springer-Verlag.
- [Farquhar et al., 1996] Farquhar, A., Fikes, R. and Rice, J. (1996). **The Ontolingua Server: A Tool for Collaborative Ontology Construction.** Knowledge Systems.
- [Farrugia, 2003] Farrugia, J. (2003). **Model-theoretic semantics for the web** 12th International Conference on World Wide Web Budapest, Hungary, ACM Press, p. 29-38.
- [Fensel, 2001] Fensel, D. (2001). **Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce,** Springer.
- [Fensel, 2002] Fensel, D. (2002). **Ontology-Based Knowledge Management** Computer Vol.(35), No. 11, p. 56-59.
- [Frakes et al., 1998] Frakes, W., Prieto-Diaz, R. and Fox, C. (1998). **DARE: Domain Analysis and Reuse Environment.** Annals of Software Engineering Vol.(5), No. 1, p. 125-141.
- [Frakes and Succi, 2001] Frakes, W. and Succi, G. (2001). **An Industrial Study of Reuse, Quality, and Productivity.** Journal of Systems and Software Vol.(57), No. 2, p. 99-106.
- [Frakes and Fox, 1995] Frakes, W. B. and Fox, C. J. (1995). **Sixteen Questions About Software Reuse.** Communications of the ACM Vol.(38), No. 6, p. 75-87.
- [Frakes and Isoda, 1994] Frakes, W. B. and Isoda, S. (1994). **Success Factors of Systematic Software Reuse.** IEEE Software Vol.(11), No. 01, p. 14-19.
- [Frakes and Pole, 1994] Frakes, W. B. and Pole, T. P. (1994). **An Empirical Study of Representation Methods for Reusable Software Components.** IEEE Transactions on Software Engineering Vol.(20), No. 8, p. 617-630.
- [Garcia et al., 2006a] Garcia, V. C., Lucrédio, D., Almeida, E. S., Fortes, R. P. M. and Meira, S. R. L. (2006a). **Towards a Code Search Engine based on the State-of-the-Art and Practice.** 13th IEEE Asia-Pacific Software Engineering Conference (APSEC), Component-Based Development Track, Bangalore, India, p. 61-68.
- [Garcia et al., 2006b] Garcia, V. C., Lucrédio, D., Durão, F. A., Santos, E. C. R., Almeida, E. S., Fortes, R. P. M. and Meira, S. R. L. (2006b). **From Specification to Experimentation: A Software Component**

- Search Engine Architecture.** The 9th International Symposium on Component-Based Software Engineering (CBSE 2006), Västerås, Sweden, Springer-Verlag, p. 82-97.
- [Glass, 2002] Glass, R. L. (2002). **Facts and Fallacies of Software Engineering**, Addison-Wesley Professional.
- [Golder and Huberman, 2006] Golder, S. and Huberman, B. A. (2006). **Usage patterns of collaborative tagging systems.** Journal of Information Science Vol.(2), No. 32, p. 198 - 208.
- [Gomez-Perez et al., 2004] Gomez-Perez, A., Corcho, O. and Fernandez-Lopez, M. (2004). **Ontological Engineering: with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web**, Springer.
- [Google, 2006] Google. (2006). **Google Code Search**, <http://www.google.com/codesearch/>. Retrieved January, 6, 2007
- [Gruber, 2002] Gruber, T. (2002). **What is an Ontology?** Retrieved January, 6, 2007, from <http://www-ksl.stanford.edu/kst/what-is-an-ontology.html>
- [Guarino, 1998] Guarino, N. (1998). **Formal Ontology and Information Systems.** International Conference on Formal Ontologies in Information Systems Trento, Italy, IOS Press, p. 3-15.
- [Guarino et al., 1999] Guarino, N., Masolo, C. and Vetere, G. (1999). **OntoSeek: Content-Based Access to the Web.** IEEE Intelligent Systems Vol.(14), No. 3, p. 70-80.
- [Harter, 1992] Harter, S. P. (1992). **Psychological relevance and information science.** Journal of American Society for Information Science Vol.(43), No. 9, p. 600-615.
- [Heineman and Council, 2001] Heineman, G. T. and Council, W. T. (2001). **Component-Based Software Engineering: Putting the Pieces Together.** USA, Addison-Wesley.
- [Henninger, 1993] Henninger, S. (1993), **Locating Relevant Examples for Example-Based Software Design**, University of Colorado, Boulder,
- [Henninger, 1994] Henninger, S. (1994). **Using Iterative Refinement to Find Reusable Software.** IEEE Software Vol.(11), No. 5, p. 48-59.
- [Henninger and Belkin, 1996] Henninger, S. and Belkin, N. J. (1996). **Interface issues and interaction strategies for information retrieval systems.** CHI '96: Conference companion on Human factors in computing systems, Vancouver, British Columbia, Canada, ACM Press, p. 352-353.
- [Holmes and Murphy, 2005] Holmes, R. and Murphy, G. C. (2005). **Using structural context to recommend source code examples.** 27th International Conference in Software Engineering, St. Louis, MO, USA, ACM Press, p. 117--125.
- [Holsapple and Joshi, 2002] Holsapple, C. W. and Joshi, K. D. (2002). **A collaborative approach to ontology design.** Communications of the ACM Vol.(45), No. 2, p. 42 - 47.
- [Hotho et al., 2006] Hotho, A., Jäschke, R., Schmitz, C. and Stumme, G. (2006). **Information Retrieval in Folksonomies: Search and Ranking.** European Semantic Web Conference, Budva, Montenegro, Springer, p. 411-426.
- [Isoda, 1995] Isoda, S. (1995). **Experiences of a Software Reuse Project.** Journal of Systems and Software Vol.(30), No. 3, p. 171-186.

-
- [J.S.Poulin, 1997] J.S.Poulin (1997). **Measuring Software Reuse**, Addison Wesley.
- [Jeffrey and Hunter, 2006] Jeffrey, S. and Hunter, J. (2006). **A Semantic Search Engine for the Storage Resource Broker**. 3rd Semantic Grid Workshop, Athens, p. 1.
- [Koders, 2006] Koders. (2006). **Koders - Source Code Search Engine**. Retrieved 06 January 2006, from <http://www.koders.com>
- [Krueger, 1992] Krueger, C. W. (1992). **Software Reuse**. ACM Computing Surveys Vol.(24), No. 02, p. 131-183.
- [Krugle, 2006] Krugle. (2006). **Krugle: Source Code Search Engine**. Retrieved January, 6, 2007, from <http://www.krugle.com>
- [Lam et al., 1999] Lam, W., Ruiz, M. E. and Srinivasan, P. (1999). **Automatic Text Categorization and Its Application to Text Retrieval**. IEEE Transactions on Software Engineering Vol.(11), No. 6, p. 865-879.
- [Lambrix and Tan, 2007] Lambrix, P. and Tan, H. (2007). **A Tool for Evaluating Ontology Alignment Strategies** Journal on Data Semantics VIII Vol.(4380/2007), No. 8, p. 182-202.
- [Lenz et al., 1987] Lenz, M., Schmid, H. A. and Peter F. Wolf, A. L. (1987). **Software reuse through building blocks**. IEEE Software Vol.(4), No. 4, p. 34-42.
- [Lim, 1994] Lim, W. C. (1994). **Effects of Reuse on Quality, Productivity and Economics**. IEEE Software Vol.(11), No. 5, p. 23-30.
- [Lim, 1998] Lim, W. C. (1998). **Managing Software Reuse**, Prentice Hall.
- [Lisboa et al., 2007] Lisboa, L. B., Garcia, V. C., Almeida, E. S. and Meira, S. R. d. L. (2007). **ToolDay A Process-Centered Domain Analysis Tool**. 21st Brazilian Symposium on Software Engineering, Tools Session. João Pessoa, Brazil.
- [Lucrédio et al., 2004] Lucrédio, D., Almeida, E. S. and Prado, A. F. (2004). **A Survey on Software Components Search and Retrieval**. 30th IEEE EUROMICRO Conference, Component-Based Software Engineering Track, Rennes - France, IEEE/CS Press, p. 152-159.
- [Lucrédio et al., 2008] Lucrédio, D., Brito, K., Garcia, V. C., Almeida, E. and Meira, S. R. d. L. (2008). **Software Reuse: The Brazilian Industry Scenario**. Journal of Systems and Software Vol.(81), No. 4.
- [Maarek et al., 1991] Maarek, Y. S., Berry, D. M. and Kaiser, G. E. (1991). **An Information Retrieval Approach for Automatically Constructing Software Libraries**. IEEE Transactions on Software Engineering Vol.(17), No. 8, p. 800-813.
- [Mascena, 2006] Mascena, J. C. C. P. (2006), **ADMIRE: Asset Development Metric-based Integrated Reuse Environment**, M.Sc. Dissertation, Federal University of Pernambuco, May.
- [McBride, 2001] McBride, B. (2001). **Jena: Implementing the RDF Model and Syntax Specification**. Semantic Web Workshop, Hewlett Packard Laboratories.
- [McBride et al., 2004] McBride, B., Brickley, D. and Guha, R. V. (2004). **RDF Vocabulary Description Language 1.0: RDF Schema**. W3C Recommendation, from <http://www.w3.org/TR/rdf-schema/>
- [McGuinness and Harmelen, 2004] McGuinness, D. L. and Harmelen, F. v. (2004). **OWL Web Ontology Language Overview**
- [McIlroy, 1968] McIlroy, M. D. (1968). **Software Engineering: Report on a conference sponsored by the NATO Science Committee**. NATO

- Software Engineering Conference, NATO Scientific Affairs Division, p. 138--155.
- [Mili et al., 1998] Mili, A., Mili, R. and Mittermeir, R. (1998). **A Survey of Software Components Storage and Retrieval**. Annals of Software Engineering. J.C. Baltzer AG, S. P. Ontario, Canada, University of Ottawa: 349--414.
- [Mili et al., 1995] Mili, H., Mili, F. and Mili, A. (1995). **Reusing Software: Issues and Research Directions**. IEEE Transactions on Software Engineering Vol.(21), No. 6, p. 528--562.
- [Missikof et al., 2002] Missikof, M., Navigli, R. and Velardi, P. (2002). **Integrated Approach to Web Ontology Learning and Engineering**. Computer Vol.(35), No. 11, p. 60-63
- [Morisio et al., 2002] Morisio, M., Ezran, M. and Tully, C. (2002). **Success and Failure Factors in Software Reuse**. IEEE Transactions on Software Engineering Vol.(28), No. 04, p. 340-357.
- [Neto et al., 2004] Neto, R. M. d. S., Lucrédio, D., Cunha, J. R. D. D., Bossonaro, A. A., Prado, A. F. d., Catarino, I. C. S. and Souza, A. M. d. (2004). **Component-Based Software Development Environment (CBDE)**. 6th ICEIS - International Conference on Enterprise Information Systems, Porto - Portugal, p. 338-343.
- [Nielson and Nielson, 1992] Nielson, H. R. and Nielson, F. (1992). **Semantics with Applications: A Formal Introduction**, Wiley.
- [Noy et al., 2001] Noy, N. F., Sintek, M., Decker, S., Crubzy, M., Ferguson, R. W. and Musen, M. A. (2001). **Creating Semantic Web Contents with Protege-200**. IEEE Intelligent Systems Vol.(16), No. 2, p. 60-71.
- [Oyama et al., 2004] Oyama, S., Kokubo, T. and Ishida, T. (2004). **Domain-Specific Web Search with Keyword Spices**. IEEE Transactions on Software Engineering Vol.(16), No. 1, p. 17-27.
- [Perry et al., 2000] Perry, D., Porter, A. and Votta, J. L. (2000). **Empirical studies of Software Engineering: A roadmap**. Proceedings of the 22nd International Conference on Software Engineering (ICSE'2000). Future of Software Engineering Track, Limerick Ireland, p. 345-355.
- [Podgurski and Pierce, 1993] Podgurski, A. and Pierce, L. (1993). **Retrieving Reusable Software By Sampling Behavior**. ACM Transactions on Software Engineering and Methodology Vol.(2), No. 3, p. 286--303.
- [Popov et al., 2004] Popov, B., Kiryakov, A., Ognyanoff, D., Manov, D. and Kirilov, A. (2004). **KIM - A semantic platform for information extraction and retrieval** Natural Language Engineering Vol.(10), No. 3-4, p. 375-392.
- [Poulin, 1997] Poulin, J. S. (1997). **Measuring Software Reuse**, Addison Wesley.
- [Prieto-Díaz, 1991] Prieto-Díaz, R. (1991). **Implementing faceted classification for software reuse**. Communications of the ACM Vol.(34), No. 5, p. 88--97.
- [Ravichandran and Rothenberger, 2003] Ravichandran, T. and Rothenberger, M. A. (2003). **Software Reuse Strategies and Component Markets**. Communications of the ACM Vol.(46), No. 8, p. 109--114.
- [Rennie, 2001] Rennie, J. (2001). **Improving multi-class text classification with naive Bayes**, M.I.T.
- [Rijsbergen, 1979] Rijsbergen, C. J. (1979). **Information Retrieval**. London, Butterworths.

-
- [Rine, 1997] Rine, D. C. (1997). **Success factors for software reuse that are applicable across Domains and businesses**. ACM Symposium on Applied Computing, San Jose, California, USA, ACM Press, p. 182--186.
- [Sametinger, 1997] Sametinger, J. (1997). **Software Engineering with Reusable Components**. Berlin Heidelberg, Springer-Verlag.
- [Santos et al., 2006] Santos, E. C. R., Durão, F. A., Martins, A. C., Mendes, R., Melo, C. d. A., Garcia, V. C., Almeida, E. S. and Meira, S. R. d. L. (2006). **Towards an Effective Context-Aware Proactive Asset Search and Retrieval Tool**. Sixth Workshop on Component-Based Development (WDBC), Recife, Brazil, p. 105--112.
- [Schmidt, 1999] Schmidt, D. (1999). **Why Software Reuse has Failed and How to Make It Work for You**. C++ Report: 46-52.
- [Seacord et al., 1998] Seacord, R. C., Hissam, S. A. and Wallnau, K. C. (1998). **Agora: A Search Engine for Software Components**. Dynamic Systems. Pittsburg, PA, CMU/SEI - Carnegie Mellon University/Software Engineering Institute.
- [Sebastiani, 2002] Sebastiani, F. (2002). **Machine learning in automated text categorization**. ACM Computing Surveys: 1-47.
- [Simperl et al., 2006] Simperl, E. P. B., Tempich, C. and Sure, Y. (2006). **ONTOCOM: A Cost Estimation Model for Ontology Engineering**. International Semantic Web Conference, p. 625-639.
- [Sommerville, 2004] Sommerville, I. (2004). **Software Reuse, in Software Engineering**.
- [Sugumaran and Storey, 2003] Sugumaran, V. and Storey, V. C. (2003). **A semantic-based approach to component retrieval** Vol.(34), No. 3, p. 8-24.
- [Sure et al., 2003] Sure, Y., Angele, J. and Staab, S. (2003). **OntoEdit: Multifaceted Inferencing for Ontology Engineering** Journal on Data Semantics Vol.(2800), No. 1, p. 128-152.
- [Thomason et al., 2000] Thomason, S., Brereton, P. and Linkman, S. (2000). **CLARiFi: An Architecture for Component Classification and Brokerage**. International Workshop on Component-Based Software Engineering (CBSE 2000). Limerick, Ireland.
- [Ugurel et al., 2002] Ugurel, S., Krovetz, R., Giles, C. L., Pennock, D. M., Glover, E. J. and Zha, H. (2002). **What is the code? Automatic Classification of Source Code Archives**. International Conference on Knowledge and Data Discovery, p. 623--638.
- [Vanderlei et al., 2007] Vanderlei, T. A., Durão, F. A., Martins, A. C., Garcia, V. C., Almeida, E. S. and Meira, S. R. L. (2007). **A Cooperative Classification Mechanism for Search and Retrieval Software Components**. 22nd Annual ACM Symposium on Applied Computing, Information Access and Retrieval (IAR) Track, Seoul, Korea, ACM Press, p. 866-871.
- [Yang and Liu, 1999] Yang, Y. and Liu, X. (1999). **A Re-Examination of Text Categorization Methods** Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval ACM Press.
- [Ye and Fischer, 2002] Ye, Y. and Fischer, G. (2002). **Supporting Reuse By Delivering Task-Relevant and Personalized Information**. ICSE

-
- 2002 - 24th International Conference on Software Engineering, Orlando, Florida, USA, p. 513--523.
- [Zaremski and Wing, 1995] Zaremski, A. M. and Wing, J. M. (1995). **Signature Matching: A Tool for Using Software Libraries**. ACM Transactions on Software Engineering and Methodology (TOSEM) Vol.(4), No. 2, p. 146--170.



ppendix

As previously stated in Chapter 6, each search mechanism had a specific filter query during the test execution. Table A1 presents 10 queries utilized during the tests for each search mechanism: keyword-based, semantic and facet-based. In addition, the objective of each query was established with the purpose of avoiding eventual ambiguity about the expected functionalities.

The choice of the queries was based on common needs shared among developers; however, it is already thought to raise run queries derived from statistic analysis from the most frequent ones.

Table A.1. Queries utilized in the evaluation

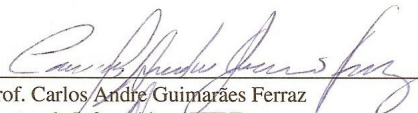
Keyword Query	Objective	Semantic Domain	Facet
file	To return source code that performs reading, writing, compressing, file stream transference and byte handling.	i/o	J2SE/infrastructure
resultSet	To return source code that performs database operations such as insert, update or select.	database	J2EE/datasource
dialog	To return source code that builds a comprehensive dialog box to be used as a user graphical interface.	gui	J2SE/interface
connection	To return source code that performs database connection and catch applicable exceptions for that.	database	J2SE/datasource
request	To return source code that handle network request and session management.	network	J2EE/web


hash	To return source code that performs any hash calculus with security purpose.	security	J2SE/infrastructure
parsing	To return source code that performs parse operation over xml files to access and manipulate its content.	xml	J2SE/ infrastructure
decode	To return source code that performs calculus for decoding messages.	math	J2SE/infrastructure
buffer	To return source code that performs buffering of file stream.	io	J2SE/infrastructure
http	To return source code that manipulates the protocol for establishing a connection and exchange information through a network.	network	J2EE/web

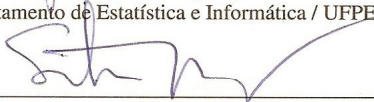
Although Table A.1 does not show the “expected results” column, this information was carefully analyzed in order to compare it with returned source codes after the test execution.

The open-source projects accessed to extract the classes which compound the dataset are: SoapUI, MvCase, Jackrabbit, Java Pilot-DB, RIFE, Protomatter, Ostermiller, Jeeves, Maracatu e Ganttproject.

Dissertação de Mestrado apresentada por **Frederico Araújo Durão** à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título “**Aplicação de uma Camada Semântica a um Engenho de Busca de Código Fonte**”, orientada pelo **Prof. Silvio Romero de Lemos Meira** e aprovada pela Banca Examinadora formada pelos professores:


Prof. Carlos André Guimarães Ferraz
Centro de Informática / UFPE


Prof. Jones Oliveira de Albuquerque
Departamento de Estatística e Informática / UFPE


Prof. Silvio Romero de Lemos Meira
Centro de Informática / UFPE

Visto e permitida a impressão.
Recife, 29 de fevereiro de 2008.


Prof. FRANCISCO DE ASSIS TENÓRIO DE CARVALHO
Coordenador da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.