

**Dean of Graduate Studies
Al-Quds University**

**Developing a Multi-Agent System for information gathering in a
distributed environment Using AUML approach**

A.Mohsen Qawasmih

M.Sc. Thesis

Jerusalem – Palestine

Rabi' Thani 1429 / April 2008

**Dean of Graduate Studies
Al-Quds University**

**Developing a Multi-Agent System for information gathering in a
distributed environment Using AUML approach**

A.Mohsen Qawasmih

M.Sc. Thesis

Jerusalem – Palestine

Rabi' Thani 1429 / April 2008

**Developing a Multi-Agent System for information gathering in a
distributed environment Using AUML approach**

**Prepared By:
A. Mohsen Khalil Qawasmih**

**B.Sc: Computer Systems Engineering – Palestine Polytechnic University -
Palestine**

Supervisor: Dr. Rushdi Hamamreh

**A Thesis Submitted in Partial fulfillment of requirements for
the degree of Master of Electronic and Computer
Engineering.**

**Department of Electrical Engineering / Master Program in
Electronic and Computer Engineering/ Al Quds University**

Rabi' Thani 1429 / April 2008

Al Quds University
Deanship of Graduate Studies
Graduate Studies / Electronic and Computer Engineering



Thesis Approval

Developing a Multi-Agent System For Information Gathering In a Distributed Environment, Using AUML Approach

Prepared By: A. Mohsen Khalil Qawasmih
Registration No: 20411051

Supervisor: Dr. Rushdi Hamamreh

Master thesis submitted and accepted, Date: 22, April 2008

The names and signatures of examining committee members are as follows:

1-Dr. Rushdi Hamamreh	Head of Committee	Signature
2-Dr. Rashid Jayousi	Internal Examiner	Signature
3-Dr. Radwan Tahboub	External Examiner	Signature

Jerusalem – Palestine
2008

Dedication:

This work is dedicated to most precious family, brothers and friends, who loved and gave me all hopefulness in life.

بسم الله الرحمن الرحيم

"قل إن صلاتي ونسكي ومحياي ومماتي لله رب العالمين، لا شريك له، بذلك أمرت


وأنا أول المسلمين"

صدق الله العظيم

Declaration:

I certify that this thesis submitted for the degree of Master is the results of my own research, except where otherwise acknowledged, and that this thesis (or any part of the same) has been submitted for a higher degree to any other university or institution.

Signed:



A. Mohsen Khalil Qawasmih

Date:- 22/04/2008

Acknowledgments

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

Thanks to "Allah" for this work.

First and foremost, I would like to thank my supervisor Dr. Rushdi Hamamreh, for his efforts, assistance, insight, patience and guidance through my graduate work.

A lot of thanks also is to my parents, specially my mother "Nadia" & father "Khalil" in addition to my lovely wife "Haya", brother "Mohannad" and sisters "Rola, Ola and Abeer" for their support and encouragement.

My great thanks is to my family, friends, and my collage (PTC-Aroub) for their help and support.

I don't forget also all the professors of Computer and Electronic Engineering department in Al-Quds University, specially, Dr. Hussein Jaddu, Dr. Labib Arafeh, Dr. Abed Al Kareem Ayad, and Dr. Ahmed Qutb. Furthermore to all Palestinians and people who loves freedom.

Abstract

The voluminous and readily available information on the Internet has given rise to exploration of intelligent technologies; for accessing, filtering, evaluating and integrating published information.

Because of the daily increase of electronically available information on the internet, an additional burden has been placed on the implementers of information gathering systems.

The set of data that represents the best response to a user's query may be the aggregation of data acquired from distributed, heterogeneous information sources. We have begun a project to build a cooperative agent for information gathering in the initial domain of computer science and medical topics.

There is a need to develop an information gathering agent to work as an advisor to his owner in a special topic. This agent should be expert in this topic and his experience is increasing with time as well as his price too.

The agent should use and make advantage from every piece of information published about the topic in the internet. His dictionary (knowledge) of this topic is re-indexed frequently with the time.

This agent should be ready all the time to give his owner the hottest documents that are much related to his topic without stopping his job in searching or any of his activities. The agent should build his own queue of targets in the internet to search and knows all the time his next step.

Our contributed system has four agent-roles:

Document Fetcher: This Agent Role uses "wget" utility for document downloading from the internet. The link of this document is taken from a storage volume which contains a queue of links to be fetched. Links queue starts from a set of start Links presented by the administrator.

Topic Analyzer: Using PLSI (probabilistic latent semantic indexing) arithmetic method. This role deals with every term and estimate's it's weight in the topic. This weight might be change up or down in the runtime of the system.

Document Evaluator: This role is responsible to calculate the relevance of the document, and the weight of it.

It contains filters with threshold to let the system decides if a document is accepted to be added to the relevant document collection or not. This filter and threshold are calculated and estimated in this thesis.

This role is also helpful in (document-ranking) phase of the system, especially after calculating the weight of each document.

Links Filter: This role has to make sure that all attracted links are useful and not repeated (already checked before) in order to increase performance.

Increasing precision and recall was our main result in this work, especially after using filters in addition to PLSI method. And it helps us a lot to select and determine the best threshold value in system's filter according to the target topic.

الملخص:-

نظراً للتزايد العظيم والكبير في حجم وكمية المعلومات المنشورة والموزعة على الشبكة العنكبوتية والحاجة الماسة لنظام بحث يستنبط ويستخرج المعلومات المفيدة والمباشرة والتي تلبي غرض المستخدم وحاجته إلى فئة معينة من المعلومات دون التشتت في جلب معلومات قد يكون جزء كبير منها غير مفيد أو غير مطلوب أصلاً.

نحن بحاجة إلى موظف أو وكيل برمجي يقوم بالعمل كمستشار يبحث عن الموضوع الذي يكلفه إياه مالكه ويقوم بجمع المعلومات والوثائق المفيدة والمرتبطة بهذا الموضوع من الانترنت دون كلل أو ملل. ويعتبر كخبير في الموضوع الذي يكلف بالبحث عنه. وخبرته تلك من المفروض أنها تزداد مع مرور الزمن وذلك يؤدي إلى زيادة ثمنه وسعره بالطبع، بحيث أن الخلفية العلمية لهذا الوكيل حول الموضوع تتطور مع كل وثيقة تدخل إلى مجموعته. أي أنه يستفيد من كل معلومة تدخل النظام.

يعمل هذا الوكيل على استنباط الموضوع المطلوب وتوزيع مفرداته ومقارنة مفردات الوثائق المحضرة بها. ويعتمد هذا الوكيل على أربعة أدوار رئيسية هي:

أولاً: جالب الوثيقة - والذي يقوم بدوره بجلب الوثيقة من الشبكة العنكبوتية بحسب عناوين محفوظة بطاوير خاص (Queue) يتم إعداده في البداية من قبل مدير النظام كنقطة انطلاق.

ثانياً: محلل الموضوع - والذي يقوم اعتماداً على نظرية الاحتمالات أنفة الذكر بفهرسة وتوزيع المفردات المتعلقة بطلب المستخدم والتي يستنبطها منه بحسب الطلبات التي قام بإدخالها وحفظها في مخزن خاص بها.

أوزان تلك المفردات من المفروض أن تتأثر مع الزمن بحيث إما أن تتعزز أو تزيد أو تنقص بحسب الوثائق التي تتجح في الوصول إلى مجموعة الوثائق المتعلقة بالموضوع المراد البحث عنه.

ثالثاً: مقيّم الوثيقة - بالاعتماد أيضاً على نظرية الاحتمالات الرياضية المذكورة أعلاه يستخدم الوكيل معادلة تم بناؤها في هذه الرسالة تجعله يبدأ بمقارنة مفردات الوثيقة ومدى تقاربها مع الموضوع الذي يستهدفه المستخدم ليعطي الوثيقة وزن نقارنه بقيمة (threshold) يتقرر بناءً عليه ما إن كانت الوثيقة مفيدة فتحفظ أم غير ذلك فتهمل.

يتم الاستفادة من هذا الوزن أيضاً في ترتيب الوثائق الناجحة للعرض بشكل تنازلي أما المستخدم من الأكثر وزناً إلى الأقل بحيث يضمن أن جميع هذه الوثائق مفيدة ومرتبطة بالموضوع ولكن تعرض حسب الأكثر ارتباطاً إلى الأقل.

رابعاً: فارز الارتباطات - في حال اجتياز وثيقة ما الاختبار بنجاح فإنه يقوم بتحليل الارتباطات الموجودة في تلك الوثيقة كل على حدا لتقدير مدى احتمالية كون هذا الارتباط مفيداً وجديراً بأن يرسل إلى الطاوير الخاص بجالب الوثائق لضمان الفعالية وعدم التكرار.

تم بحمد الله برمجة وتطبيق العناصر الأساسية لهذا النظام باستخدام لغة جافا وقد كانت له نتائج مرضية خاصة في رفع مستوى الدقة (Precision) والكفاءة والقدرة على التذكر (Recall). وذلك بمساعدة عناصر الفرز والتصفية التي تم تطوير النظام بواسطتها والقدرة على اختيار أفضل قيمة اختبار (threshold) حسب الموضوع. و سيتم عرضها في الفصول القادمة من هذه الرسالة.

TABLE OF CONTENTS

Dedication	I
Declaration	II

Acknowledgments	III
Abstract	IV
TABLE OF CONTENTS	VII
List of Figures	IX
List of Tables	XI
List of Acronyms and Abbreviations	IV
Chapter 1	
Introduction	
1.1 Information Retrieval	1
1.2	3
1.3 Thesis Motivation	4
1.4 Thesis Outline	5
Chapter 2	
Analysis of existing Algorithms and architectures	
2.1 Introduction	6
2.2 Programming models analysis.	7
2.2.1 Traditional Programming.	8
2.2.2 OOP & AOP.	9
2.2.3 UML & AUML.	10
2.3 Arithmetic IR Algorithms analysis.	12
2.3.1 Vector space algorithm.	14
2.3.2 LSI & Probabilistic IR.	17
2.3.3 Probabilistic Latent Semantic Indexing (PLSI).	21
2.4 Summary & Conclusion	23
Chapter 3	
Design of Agent architecture	
3.1 Introduction	24
3.2 Agent Architecture	25
3.2.1 Role A (Document Fetcher)	26
3.2.2 Role B (Topic Analyzer)	28
3.2.3 Role C (Document Evaluator)	29
3.2.4 Role D (Links Filter)	30
3.3 Summary & Conclusion	31
Chapter 4	
Development Agent Algorithm	
4.1 Introduction	32
4.2 PLSI Method	34
4.3 How it works?	37
4.4 Summary & Conclusion	38
Chapter 5	
Implementation and testing	
5.1 Introduction	39
5.2 Agent Major Classes	41
5.2.1 Main Class.	43
5.2.2 WGET Class.	45
5.2.3 Stemming Class.	46
5.2.4 PLSI Class.	49
5.2.5 Get Links Class.	50
5.3 Testing Experiments	51
5.3.1 Experiment 1	57

5.3.2 Experiment 2	61
5.3.3 Experiment 3	65
5.4 Summary & Conclusion	66
Chapter 6	
Conclusions and Future works	
6.1 Conclusions.	69
6.2 Future works	72
References	73
Intelligent Focused Information Agent (Published Paper) + Annex	1

List of Figures

Figure 1.1	system environment	3
Figure 2.1	Levels of abstraction that affect programming model	7
Figure 2.2	Three Agent's Roles of the system	11

Figure 2.3	Precision & Recall	12
Figure 2.4	Vector Space Model	15
Figure 2.5	Training data for Probabilistic Retrieval	18
Figure 2.6	Precision-recall curves for the 4 test collections with term weighting by PLSI compare to other methods	21
Figure 3.1	Agent Architecture AUML	27
Figure 5.1	Agent-User Interface	41
Figure 5.2	Agent's Interface with related classes	42
Figure 5.3	InterFace1 Class	43
Figure 5.4	AgentA Class	43
Figure 5.5	WGET Class	45
Figure 5.6	Converting full text to indexed terms through stemming	46
Figure 5.7	Stemmer Class	47
Figure 5.8	Experiment 1 results chat. (traditional system)	66
Figure 5.9	Experiment 2 results chat. (Our System)	67
Figure 5.10	Experiment 3 Precision & Recall on Medical topic	68
Figure 6.1	Surface chart for best threshold	71
Figure 6.2	Precision according to threshold τ	71
Figure 7.1	AgentA Class (user interface)	75
Figure 7.2	Agent's Main Method	75
Figure 7.3	Interface1 class (Frame constructor)	76
Figure 7.4	Dealing with user input	76
Figure 7.5	Creating query table from user's input field	77
Figure 7.6	CreatOutput() method	77
Figure 7.7	QcreatOutput() method	78
Figure 7.8	CcreatOutput() method	78
Figure 7.9	Component initialization	79
Figure 7.10	Bringing a document from internet	79
Figure 7.11	String's stemming before creating the table	80
Figure 7.12	WGET Class (Java code)	81
Figure 7.13	CreateAFile method.	82
Figure 7.14	Stemmer class's declaration	82
Figure 7.15	Add() method1 in stemmer class	82
Figure 7.16	Add() method2 in stemmer class	83
Figure 7.17	ToString() method in stemmer class	83
Figure 7.18	GetResultLength() method in stemmer class	83
Figure 7.19	GetResultBuffer() method in stemmer class	83
Figure 7.20	Consonant cases method in stemmer class	84
Figure 7.21	Consonants counter method in stemmer class	84
Figure 7.22	Vowel checker method in stemmer class	85
Figure 7.23	Double consonant checker in stemmer class	85
Figure 7.24	An "e" restore method in stemmer class	85
Figure 7.25	String's end checker method in stemmer class	85
Figure 7.26	Setto() method in stemmer class	86
Figure 7.27	String creator r() method in stemmer class	86
Figure 7.28	Step1 in stemming process	87
Figure 7.29	Step2 in stemming process	88
Figure 7.30	Step3 in stemming process	88
Figure 7.31	Step4 in stemming process	89

Figure 7.32	Step5 in stemming process	90
Figure 7.33	Step6 in stemming process	91
Figure 7.34	Stem() method in stemmer class	91
Figure 7.35	PLSI implementation "Mex_EMstep.c".	92
Figure 7.36	PLSI implementation "mex_Pw_d.c"	93
Figure 7.37	PLSI implementation "mex_logL.c"	94
Figure 7.38	Getlinks class	95
Figure 7.39	Link Recognizer	96

List of Tables

Differentiation between agents and objects at the programming	9
---	---

Table 2.1	level	
Table 2.2	The relation between programming paradigms, modeling techniques and programming languages	10
Table 5.1	Relevance of Test Document Collection	52
Table 5.2	Document[1..32]-Term matrix with P(w z)	53
Table 5.3	Document[33..64]-Term matrix with P(w z)	53
Table 5.4	Document[65..96]-Term matrix with P(w z)	54
Table 5.5	Document[97..128]-Term matrix with P(w z)	54
Table 5.6	Weighted terms and document [1..32] in PLSI	55
Table 5.7	Weighted terms and document [33..64] in PLSI	55
Table 5.8	Weighted terms and document [65..96] in PLSI	56
Table 5.9	Weighted terms and document [97..128] in PLSI	56
Table 5.10	Experiment 1 results (Precision & Recall) WGET[1..32]	57
Table 5.11	Experiment 1 results (Precision & Recall) WGET[33..64]	58
Table 5.12	Experiment 1 results (Precision & Recall) WGET[65..96]	59
Table 5.13	Experiment 1 results (Precision & Recall) WGET[97..128]	60
Table 5.14	Experiment 2 results (Precision & Recall) WGET[1..50]	62
Table 5.15	Experiment 2 results (Precision & Recall) WGET[51..76]	63
Table 5.16	Experiment 3 results (Precision & Recall) WGET[1..23]	64

List of Acronyms and Abbreviations

AOP	Agent Oriented Programming
AOSE	Agent-Oriented Software Engineering

AUML	Agent UML
DAI	Distributed Artificial Intelligence
DF	Document Frequency
GF	Global frequency
IDF	Inverse Document Frequency
IR	Information Retrieval
IWS	Internet world stats
JADE	Java Agent Development
LF	Local frequency
LSI	Latent Semantic Indexing
MAS	Multi-Agent System
OOP	Object Oriented Programming
PLSI	Probabilistic Latent Semantic Indexing
PRP	Probability Ranking Principle
SC	Similarity Coefficient
SVD	Singular Value Decomposition
TF	Term Frequency
UML	Unified Modeling language
VSM	Vector Space Model

Chapter 1

Introduction

1.1 Information Retrieval

A continuous growth of the internet usage with billions of published documents and data distributed around the world, demands a useful, fast, accurate and intelligent search system that satisfies users needs and queries. We noticed from IWS (Internet world statistics) website[9], the rapid growth specially the last few years. We can see on Mar-2007 that the internet population grows up to 6,574,666,417 documents. while the internet latest data usage was about 1,114,274,426 documents. This indicates the huge amount of document that is distributed around the world [9].

It's well known that search engines with centralized architecture can't index the whole Internet because the exponential growth of the number of documents published in the Internet. Search engine with distributed architecture is scalable solution of this problem [3].

Our system is based on Intelligent Information gathering Agent, aims to help the user, and its main roles are melting together to help the topic searcher to find his targets in an open system that contains other multi-agent system (MAS) and many other information access technologies, by investigating to what extent methods from IR mathematical methods, Intelligent search systems.

Data storage Systems and Information Retrieval (IR) can be applied to information discovery by themes of information agents in the Internet and the World Wide Web[8], [1].

In the framework of our suggested architecture, we use a set of topic target collections of electronic documents published in the Internet. These collections belong to different owners who are responsible for their content, indexing and quality of search. Administrator's demand is automatically propagated to one or more collections with topics relevant to his target topic [1], [2], [4].

This thesis describes architecture of an autonomous agent that gathers information from distributed environment, as Internet, to build subject-specific collection, and to extract information from documents using probabilistic latent semantic indexing algorithm.

It also describes techniques for developing distributed and adaptive agent that coordinate to retrieve, filter and recommend information relevant to the owner, from various web sources. The knowledge of agent based on semantic indexing by analyzing multiple topics in HTML pages, with the help of probability mathematical method which is called PLSI.

In contrast to most current research that has been investigated single-agent approaches, we are developing an agent with a collection of four major roles that team up on demand, depending on the user's query, topic and links queue, to access, filter and integrate information distributed in the internet.

We are investigating techniques for developing distributed adaptive roles of an information agent that coordinate to retrieve, filter and fuse information relevant to the user's query, topic and links queue, as well as anticipate user's information needs.

In our system, information gathering is seamlessly integrated with search support. In this thesis we present the distributed system architecture, agent collaboration interactions, and a reusable set of software components for structuring agents.

We have implemented most of this system framework and get successful output results compared to similar traditional systems. By developing collaborating agent's roles, using JAVA language, in diverse complex real world tasks, such as organizational document searching, and topic indexing management.

1.2 Agent theory.

An agent is a computational entity such as a software program or a crawler[2] that can be viewed as perceiving and acting upon its environment and that is autonomous in that its behavior at least partially depends on its own experience. As an intelligent entity, an agent operates flexibly and rationally in a variety of environmental circumstances given its perceptual and effectual equipment. Behavioral flexibility and rationality are achieved by an agent on the basis of key processes such as problem solving, planning, decision-making, and learning. [11].

The study of multi-agent systems began in the field of distributed artificial intelligence (DAI) about 20 years ago. Today these systems are not simply a research topic, but are also beginning to become an important subject of academic teaching and industrial and commercial application [5], [7].

Our agent is expected to establish new cooperation among research groups in the related areas mentioned above, but also to strengthen existing contacts and focus scattered efforts for research on and development of intelligent information agents.

In particular, managing and controlling such networks, the services provided, and the communications involved, is crucial to keep Internet a useful tool in the future. However, there is a growing awareness that current centralized IR architectures will soon reach the limits of their scalability. Some Scientifics argue that distributed but coordinated mechanisms that support adaptation and self-optimization of Information Agent societies can be an answer to this problem. [8],[4].

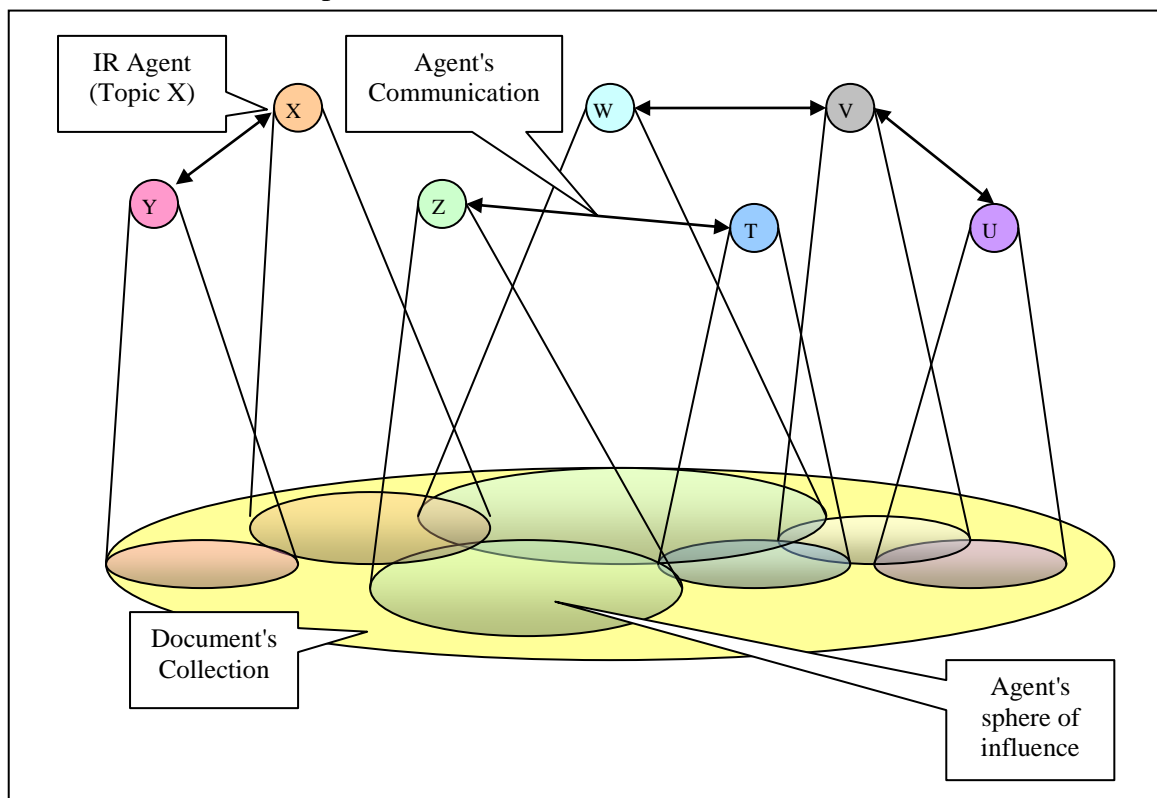


Figure 1.1 system environment

1.3 Thesis Motivation.

Recent developments in agent-based computing and software engineering have revealed a significant potential and urgent demand for a close interaction among these disciplines.

On the one hand, as the number of special-topic agent-based software systems grows it becomes important to build IR systems that use software engineering technology that is specifically tailored for agent systems.

Thus software engineering is crucial to the textual and search application success of agent-based computing.

On the other hand, as today's and tomorrow's standard software systems are required to operate in increasingly complex – distributed, large, open, dynamic, unpredictable, heterogeneous and highly interactive – IR application environments, it appears to be very promising and natural to build these systems in terms of agent and multi-agent technology.

Thus agent orientation can serve as a useful paradigm in software engineering. The field emerging as a result of this mutual demand for interaction has been referred to as Agent-Oriented Software Engineering (AOSE)[12].

1.4 Thesis Outline

In the next chapter, we are going to analyze existed Algorithms and architectures that are related to such systems.

We have analyzed existed programming models such as traditional programming in addition to OOP & AOP. And programming approaches like UML & AUML with its latest progresses and developments.

Also, we have analyzed existed arithmetic IR Algorithms that is related to our agent system, like Vector space algorithm, LSI and Probabilistic models. And justify why we have chosen PLSI model in our agent.

Chapter three introduces the design of agent architecture. Using AUML including its main four major roles which are, Role A (Document Fetcher), Role B (Topic Analyzer), Role C (Document Evaluator), and Role D (Links Filter).

And we will discuss the major job of each role and how it works according to its position in the system.

In chapter four we are going to discuss in details the development of the agent algorithm which depends on PLSI methods and how it works in our agent system. We will present the mathematical functions we used in it.

The implementation of our agent will be presented in chapter five. Using JAVA programming language we are going to preview our agent's major classes starting from main class, moving to WGET class, stemming class, Getlinks class and the arithmetic filter of the system which is PLSI class.

Two experiments were applied to our agent system, concentrating on one topic. The first experiment used traditional PLSI method without our contributed filter, while the other experiment was with the filter. Then we got results from each one and note the difference between them.

These results were analyzed, summarized and presented in the chapter six. Then, our conclusions and further work presented too.

By the end of the thesis, references with our published paper including Java code in the appendix were presented.

Chapter 2

Analysis of existing Algorithms and architectures

2.1 Introduction

Agents can be defined in many ways and there is no one universally correct or acceptable definition. An agent can be defined (simply) as an autonomous entity that can sense and act upon its environment. Such simple agents are analogous but not equivalent to orthodox automata [28]. More sophisticated descriptions draw on concepts such as intentionality, social ability, adaptation, learning, communication etc. Agent environments vary considerably, from synthetic worlds to those which robots inhabit through to more abstract worlds consisting of information and knowledge [11].

Also, agents are a very promising technology for information retrieval. Some applications are intelligent IR interfaces, mediated searching and brokering, and clustering and categorization. An agent-based approach means that IR systems can be more scalable, flexible, extensible, and interoperable, using agents that route information, broker requests, and share metadata.

The architecture and composition of an agent typically reflects its environment and the role(s) it plays within that environment; i.e. the challenge of an agents problem or niche space. Our current research relies heavily on the concept of agency across a number of different domains, and categories of processes, agencies and agents. For example, we can consider our ongoing research into decision support systems as that of an investigation into tightly-coupled agent communities making use of modern but relatively orthodox AI techniques [28].

One of the threads that draw this work together is that of investigating computational architectures that allow or help to support computational intelligent methods on IR applications.

2.2 Programming models analysis.

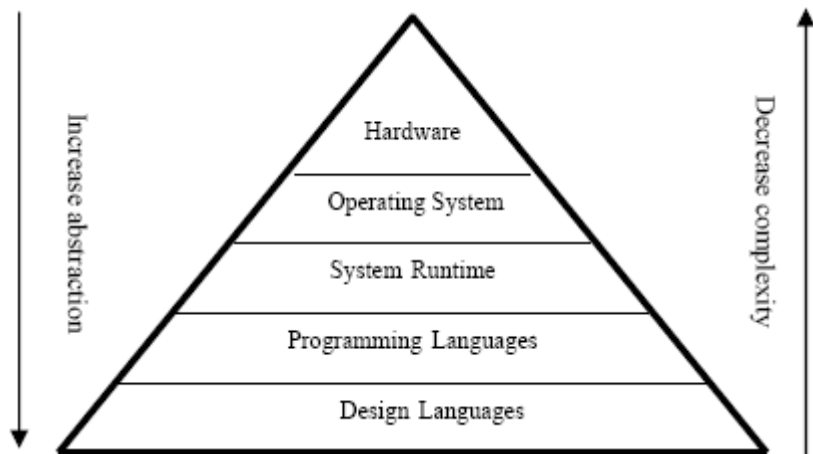


Figure 2.1: Levels of abstraction that affect programming model

Programming models are affected by different aspects at different level of abstraction. Figure 2.1 shows these levels of computing abstraction. The triangle shape shows the abstraction level that dominates the others. For instance, hardware plays a dominant factor on operating systems, while operating systems abstract factors related to hardware, and so on[11].

2.2.1 Traditional Programming.

We can note the differences between programming models with the traditional programming in the runtime system which provides the environment for program interpretation. These environments can be radically different between different paradigms.

These environments may be restricted to administrative tasks or they may also provide slightly more elaborate services. At this level of abstraction, “agents” have distinct behavior from “objects”. In an object-oriented runtime system, the objects are statically represented by the objects’ architecture.

This architecture contains the current state of any object and objects’ relations to the object classes, which subsequently determine the operations that can be performed by this object. An object is usually represented as a collection of data elements with associated functions and the granularity of objects is potentially not limited.

The object management system is responsible for managing the relations between objects and classes (e.g. the inheritance relation) and for the manipulation of objects (e.g. objects creation or destruction). Furthermore, the object management system is also responsible for dynamic aspects, such as method selection of polymorphous objects, exception handling and garbage collection.

In an agent-oriented runtime system, things are distinctly more complicated. Agent architectures are far more complex than the object architecture, especially because of the dynamic aspects that agents deal with. Each agent perceives the state of its environment, integrates the perceived facts in its knowledge base, forms beliefs, desires, goals and intentions to act and finally executes the planned activities (possibly in coordination with other agents)[11].

2.2.2 OOP & AOP.

To sense differences between OOP & AOP at programming language level of abstraction, the syntax and semantics of a programming language for the manipulation of entities at the system runtime level is defined.

The programs that are written in a particular programming language are interpreted at the system run-time level. In the programming language level, as well as it is at the runtime level, there is a differentiation between objects and agents, as it is shown in the following table (Table:2.1)

Table 2.1: Differentiation between agents and objects at the programming level[11]

	OOP	AOP
Structural Elements	Abstract class	Generic role
	Class	Domain specific role
	Member variable	Knowledge, belief
	Method	Capabilities, (complex and primitive) actions
Relation & Communication	Collaboration(uses)	Negotiation
	Composition (has)	Institutionalized agents, groups of agents
	inheritance (is)	Role multiplicity
	Instantiation	Domain specific role and individual knowledge
	Polymorphism	Service matching

2.2.3 UML & AUML.

Design languages are further abstractions from a particular programming language that aim at the conceptual modeling of a system at a more coarse grained level. Design languages often use graphical notations that make it easier for the designer to use and manipulate the overall system structure.

In the object oriented community UML is a well established design language being supported by case tools such as the Rational Rose Software®, which can transform the object architecture into class code in Java or C++, in conjunction to other useful design utilities.

Table 2.2: The relation between programming paradigms, modeling techniques and programming languages (updated from [11])

Programming languages, and Software Modeling Techniques				
	Programming language	Analysis	Design	
Programming Paradigms	Top down (Monolithic)	Assembly High level language	Textual, Algorithms Flowcharts Algorithms	
	Structural (Modular)	High level languages with built-in support routines	Dataflow diagram HIPO Chart Data Structure Diagram and Structure Chart	
	Object Oriented	Object Oriented high-level languages, Object support libraries.	UML: Use Case & Collaboration Diagrams	UML: Class diagrams and its relations, State Machine...
	Agent Oriented	Agent Platform, an Agent Oriented Language does not exist yet.	AUML: Use Case & Collaboration Diagrams	AUML Class diagrams and its relations, State Machine...

In the agent-based world there is no uniform design language mainly due to the ellipsis of an agent oriented programming paradigm. However, there is a large number of design toolkits for special kind of agent architectures and platforms. But lately AUML is starting to be a new step toward a complete design language as well as UML As shown in Table 2.2.

To know the goal behind moving from UML to AUML we should understand that Multi-agent systems (MAS) are often characterized as extensions of object-oriented systems. So, this overly simplified view has often troubled system designers as they try to capture the unique features of MAS systems using "object oriented" tools. In response, an agent-based unified modeling language (AUML) is being developed.

Instead of reliance on the UML, we used AUML in our system which is based on IR (Information retrieval) agent where it makes sense. We do not want to be restricted by UML; we only want to capitalize on it where we can. The general philosophy, then, is: "When it makes sense to reuse portions of UML, then do it; when it doesn't make sense to use UML, use something else or create something new." [11]

Since generating agents come as part of the software engineering process, they have to be consistent and complement other models built during this process. This system shows how our agent can be integrated within UML towards AUML.

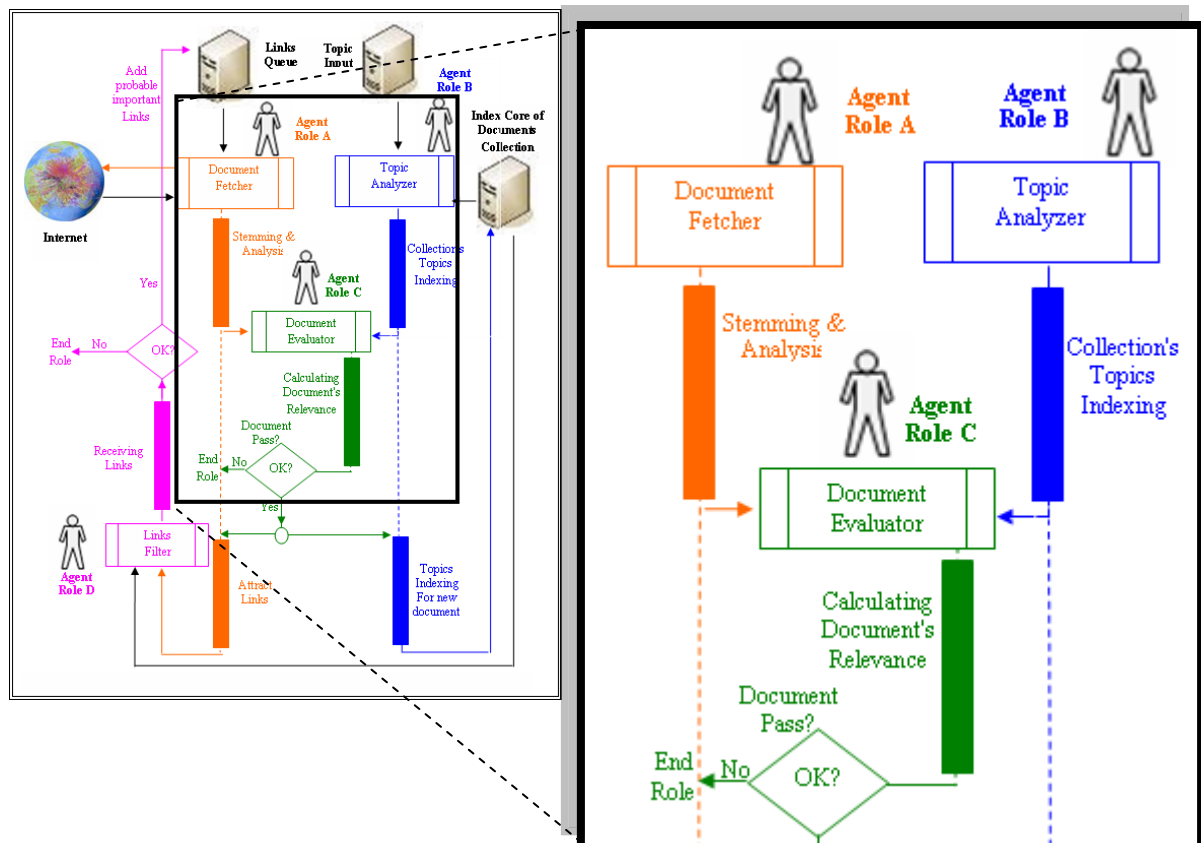


Figure 2.2 Three Agent's Roles of the system

Our agent roles are integrated with other UML diagrams for specifying agents' interaction. Returning to our Agent Architecture, let us consider the specification of the roles functionality. Starting from the UML Use Agent's roles diagram in Figure 2.2, interactions among interdependent roles are specified by means of AUML. So we can specify the agent's roles given in Figure 3.1 that represents the interaction among agent's roles communicating with each other. Which shows a combination of simple, agent instantiation, and spanning role couples.

This new development on Agent view can be integrated with UML, reaching to AUML that supports object classes, as well as agent and role classes. By this way an agent entity is free from any role "burden", it can move from one role to another without any pre-assigned agent-role mapping, agent entities can be instantiated to perform atomic roles, agents can move freely and be instantiated according to system functionality constrains (agent –role switching constrains) or according to agents' internal state[10].

2.3 Arithmetic IR Algorithms analysis.

Our agent target is focused on Information Retrieval (IR) which is devoted to finding relevant documents, not finding simple matches to patterns. Yet, often when information retrieval systems are evaluated, they are found to miss numerous relevant documents. Moreover, users have become complacent in their expectation of accuracy of information retrieval systems.

We'll show the critical document categories that correspond to an issued topic. Namely, in the collection there are documents which are retrieved, and there are those documents that are relevant. In a perfect system, these two sets would be equivalent; we would only retrieve relevant documents.

In reality, systems retrieve many non-relevant documents. To measure effectiveness, two ratios are used: *precision* and *recall*.

Precision is the ratio of the number of relevant documents retrieved to the total number retrieved. Precision provides an indication of the quality of the answer set. However this dose not considers the total number of relevant documents. A system might have a good precision by retrieving ten documents and finding that nine are relevant (a 0.9 precision), but the total number of relevant documents is also important. If there were only nine relevant documents, the system would be a huge success, however if millions of documents were relevant and desired, this would not be a good result set[1].

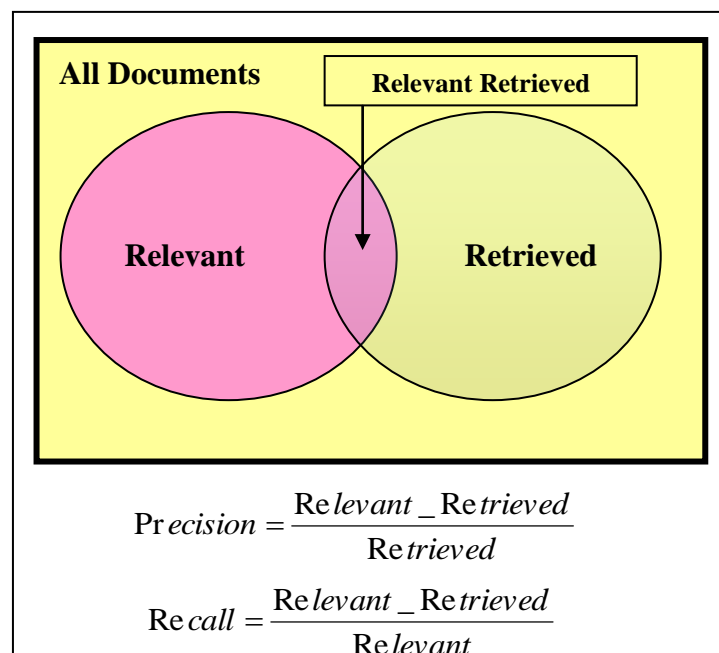


Figure 2.3 Precision & Recall

Recall considers the total number of relevant documents; it is the ratio of the number of relevant documents retrieved to the total number of documents in the collection that are believed to be relevant. Computing the total number of relevant documents is non-trivial. The only sure means of doing this is to read the entire document collection. Since this clearly not feasible, an approximation of the number is obtained.

Next we will focus on the current strategies to find relevant documents quickly. The quest to find efficient and effective IR algorithm continues as well as on our agent.

A retrieval strategy is an algorithm that takes a query Q and a set of documents D_1, D_2, \dots, D_n and identifies the Similarity Coefficient $SC(Q, D_i)$ for each of the documents $1 \leq i \leq n$. We will focus on the following arithmetic algorithms:-

- Vector Space Model (VSM) – Both the query and each document are represented as vectors in the term space. A measure of similarity between the two vectors is computed.
- Latent Semantic Indexing – The occurrence of terms in the documents is represented with a term document matrix. The matrix is reduced via Singular Value Decomposition (SVD) to filter out the noise found in a document so that two documents which have the same semantic are located close to one another in a multidimensional space.
- Probabilistic Retrieval – A probability based on the chance that a term will appear in a relevant document is computed for each term in the collection. For terms that match between a query and a document, the similarity measure is computed as the combination of the probabilities of each matching terms.

2.3.1 Vector space algorithm.

A vector space model computes a measure of similarity by defining a vector that represents the query. The model is based on the idea that, in some rough sense, the meaning of a document is conveyed by the words used. If one can represent the words in the document by a vector, it is possible to compare documents with queries to determine how similar their content is.

If a query is considered to be like a document, a Similarity Coefficient (SC) that measures the similarity between a document and a query can be computed. Documents whose content, as measured by the terms in the document, correspond most closely to the content of the query are judged to be the most relevant. Figure 2.4 shows the basic notion of the vector space model in which vectors are that represents a query and multiple documents are shown.

This model involves constructing a vector that represents the terms in the document and another vector that represents the terms in the query. Then, a method should be chosen to measure the closeness of any document vector to the query vector.

One could look at the magnitude of the difference vector between two vectors, but this would tend to make any large document appear to be not relevant to most queries. Which typically are short.

The traditional method of determining closeness of two vectors is to use the size of the angle between them. This angle is computed by using the inner product (or dot product); however, it is not necessary to use the actual angle. Any monotonic function of the angle suffices. Often SC is used instead of an angle. Computing this number is done in a variety of ways, but the inner product generally plays a prominent role.

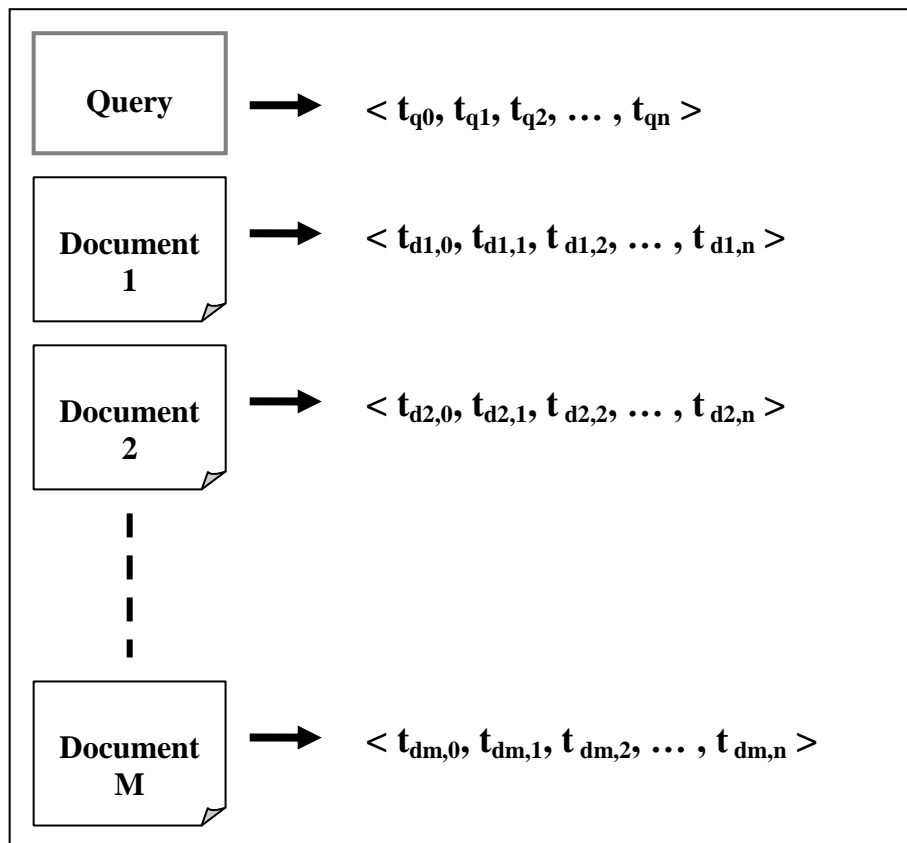


Figure 2.4 Vector Space Model

To construct a vector that corresponds to each document, consider the following definitions:-

- t = number of distinct terms in the document collection
- tf_{ij} = number of occurrences of term t_j in document D_i . This referred to as the term frequency.
- df_j = number of documents which contains t_j . This is the document frequency.
- $idf_j = \log \left(\frac{d}{df_j} \right)$ where d is the total number of documents. This is the inverse df .

The vector of each documents has n components and contains an entry for each distinct term in the entire document collection. The components in the vector are filled with weights computed for each term in the document collection. The terms in each document are automatically assigned weights based on how frequently they occur in the entire document collection and how often a term appears in a particular document. The weight of a term in a document increases the more often the term appears in one document and decrease the more often it appears in all other documents.

A weight computed for a term in a document vector is non-zero only if the term appears in the documents. For large document collection consisting of numerous small documents, the document vectors are likely to contain mostly zeros.

The weighting factor for a term in a document is defined as a combination of term frequency, and inverse document frequency. That is, to compute the value of the j th entry in the vector corresponding to document i , the following equation is used [26]:

$$d_{ij} = tf_{ij} \times idf_j$$

When a document retrieval system is used to query a collection of documents with t distinct collection-wide terms, the system computes a vector $D (d_{i1}, d_{i2}, \dots , d_{it})$ of size t for each document. The vectors are filled with term weights as described before. Similarly, a vector $Q (w_{q1}, w_{q2}, \dots , w_{qt})$ is constructed from terms found in the query.

A simple SC between a query Q and a document D_i is defined by the dot product of two vectors. Since a query vector is similar in length to a document vector, this same measure is often used to compute the similarity between two documents [26].

$$SC(Q,D_i) = \sum_{j=1}^t w_{qj} \times d_{ij}$$

2.3.2 LSI & Probabilistic IR.

Latent Semantic Indexing (LSI) – Matrix computation is used as a basis for information retrieval in the retrieval strategy called LSI. The premise is that more conventional retrieval strategies like VSM all have problems because they match directly on keywords. Since the same concept can be described using many different keywords, this type of matching is prone to failure. The authors cite a study in which two people used the same word for the same concept only twenty percent of the time[26].

Searching for something that is closer to representing the underlying semantic of the document is not a new goal. Applied here, the idea is not to find a canonical knowledge presentation, but to use a matrix computation, in particular Singular Value Decomposition (SVD). This filters out the noise found in a document, such that two documents that have the same semantic (weather or not they have the same matching terms) will be located close to one another in a multi-dimensional space.

The process is relatively straightforward. A term document matrix A is constructed such that location (i,j) indicates the number of times term i appears in document j . A SVD of this matrix results in matrices $U \Sigma V^T$ such that Σ is a diagonal matrix. U is a matrix represents each term in a row. Each column of A represents documents. The values in Σ are referred to as the singular values. The singular value can be then stored by magnitude and the top k values are selected as a means of developing a "latent Semantic" representation of the A matrix. The remaining singular values are then set to 0. Only the first k columns are kept in U_k ; only the first k rows are recorded in a V_k^T . After setting the results to 0, a new A' matrix is generated to approximate $A = U \Sigma V^T$ [26].

Comparison of two terms is done via an inner product of the two corresponding rows in U_k . Comparison of two documents is done as an inner product of two corresponding rows in V_k^T . A query-document similarity coefficient treats the query as a document and computes the SVD. However, the SVD is computationally expensive; so, it is not recommended that this be done as a solution. Techniques that approximate Σ and avoid the overhead of the SVD exist. For infrequently updated document collection, it is often pragmatic to periodically compute the SVD.

The Probabilistic model computes the similarity coefficient (SC) between a query and a document as the probability that the document will be relevant to the query. This reduces the relevance ranking problem to an application of probability theory.

Probability theory can be used to compute a measure of relevance between query and a document. All of the work on probabilistic retrieval stems from the concept of estimating a term's weight based on how often the term appears or doesn't appear in relevant documents and non-relevant documents, respectively.

Simple Term Weights – The use of term weights is based on the probability ranking principle (PRP), which assumes that optimal effectiveness occurs when documents are ranked based on an estimate of the probability of their relevance to a query. The key is to assign probabilities to components of the query and then use each of these as evidence in computing the final probability that a document is relevant to a query.

The terms in the query are assigned weights which correspond to the probability that a particular term, in a match with a given query, will retrieve a relevant document. The weights for each term in the query are combined to obtain a final measure of relevance [26].

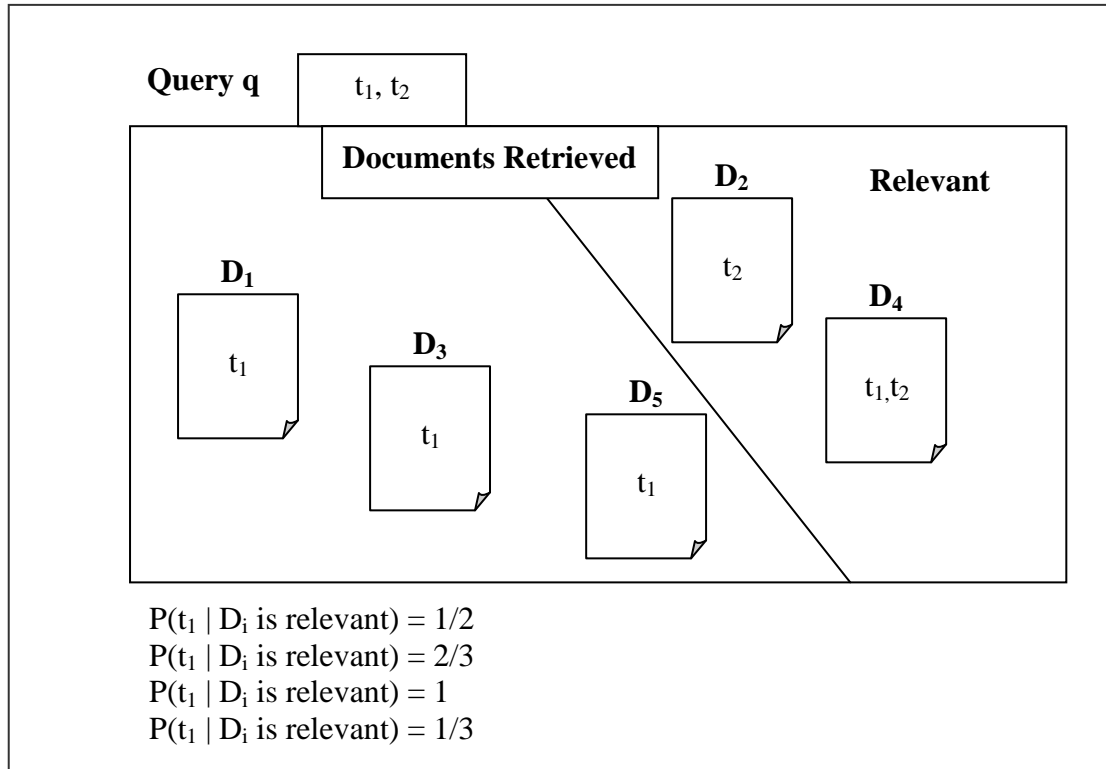


Figure 2.5 Training data for Probabilistic Retrieval

The terms in the query can be viewed as indicators that a given document is relevant. The presence or absence of a query term A can be used to predict whether or not a document is relevant. Hence, after a period of observation, it is found that when term A is in both the query and the document, there is an x percent chance the document is relevant. We then assign a probability to term A . Assuming independence of terms, this can be done for each of the terms in the query. Ultimately, the product of all the weights can be used to compute the probability of relevance.

In Figure 2.5, we will show the need for training data with most probabilistic models. A query with two terms, t_1 and t_2 , is executed. Five documents are returned and an assessment is made that the documents two and four are relevant. From this assessment, the probability that a document is relevant (or non-relevant) given that it contains term t_1 is computed. Likewise, the same probabilities are computed for t_2 . Clearly these probabilities are estimates based on training data. The idea is that sufficient training data can be obtained so that when a user issues a query, a good estimate of which document are relevant to the query can be obtained.

Consider a document, d_i , consisting of t terms (w_1, w_2, \dots, w_t) , where w_i is the estimate that term i will result in this document being relevant the weight or "odds" that document d_i is relevant is based on the probability of relevance of each term in the document. For a given

term in a document, its contribution to estimate of relevance for the entire document is computed as:

$$\frac{P(w_i | rel)}{P(w_i | nonrel)}$$

Given our independence assumption, we can multiply the odds for each term in a document to obtain the odds that the document is relevant. Taking the log of the product yields:

$$\log\left(\prod_{i=1}^t \frac{P(w_i | rel)}{P(w_i | nonrel)}\right) = \sum_{i=1}^t \log\left(\frac{P(w_i | rel)}{P(w_i | nonrel)}\right)$$

We note that these values are computed based on the assumption that terms will independently in the relevant and non-relevant documents. The assumption is also made that if one term appears in a document, then it has no impact on whether or not another term will appear in the same document.

The means of estimating the individual term weights by the following two assumptions[26]:

I1: The distribution of terms in relevant documents is independent and their distribution in all documents is independent. It indicates that terms occur randomly within a document – that is, the presence of one term in a document is no way impact the presence of another term in the same document. This states that distribution of terms across all documents is independent unconditionally for all documents – that is, the presence of one term in a document is no way impacts the presence of the same term in other documents.

I2: The distribution of terms in relevant document is independent and their distribution in non-relevant documents is independent. It indicates that terms in relevant documents are independent – that is, they satisfy I1 and terms in non-relevant documents also satisfy I1.

They also presented to methods, referred to as ordering principles, for presenting the result set:-

O1: Probable relevance is based only on the presence of search terms in the documents. It indicates that documents should be highly ranked only if they contain matching terms in the query (i.e., the only evidence used in which query terms are actually present in the term.

O2: Probable relevance is based on both the presence of search terms in documents and their absence from documents.

Four weights are then derived based on different combination of theses ordering principles and independence assumptions. Given term, t , consider the following quantities:

- N = number of documents in the collection.
- R = number of relevant documents from a given query q .
- n = number of documents that contain term t .

- r = number of relevant documents that contains term t .

Choosing I1 and O1 yields the following weight:

$$w1 = \log \left(\frac{\frac{r}{R}}{\frac{n}{N}} \right)$$

Choosing I2 and O1 yields the following weight:

$$w2 = \log \left(\frac{\frac{r}{R}}{\frac{n-r}{N-R}} \right)$$

Choosing I1 and O2 yields the following weight:

$$w3 = \log \left(\frac{\frac{r}{R-r}}{\frac{n}{N-n}} \right)$$

Choosing I2 and O2 yields the following weight:

$$w4 = \log \left(\frac{\frac{\frac{r}{R-r}}{n-r}}{(N-n)-(R-r)} \right)$$

The claimed advantage to the probabilistic model is that it is entirely based on probability theory. The implication is that other models have a certain arbitrary characteristics. They might perform well experimentally, but they lack a sound theoretical basis because the parameters are not easy to estimate. Either complete training data are required, or an inaccurate estimate must be made[26].

2.3.3 Probabilistic Latent Semantic Indexing (PLSI).

Probabilistic Latent Semantic Indexing is a novel approach to automated document indexing which is based on a statistical latent class model for factor analysis of count data. Fitted from a training corpus of text documents by a generalization of the Expectation Maximization algorithm, the utilized model is able to deal with domain-specific synonymy as well as with polysemous words.

In contrast to standard Latent Semantic Indexing (LSI) by Singular Value Decomposition, the probabilistic variant has a solid statistical foundation and defines a proper generative data model.

Retrieval experiments on a number of test collections indicate substantial performance gains over direct term matching methods as well as over LSI. In particular, the combination of models with different dimensionalities has proven to be advantageous [27].

Compared to standard latent semantic analysis which stems from linear algebra and downsizes the occurrence tables (usually via singular value decomposition), probabilistic latent semantic analysis is based on a mixture decomposition derived from a latent class model. This results in a more principled approach which has a solid foundation in statistics.

Considering observations in the form of co-occurrences (w,d) of words and documents, PLSI models the probability of each co-occurrence as a mixture of conditionally independent multinomial distributions:

$$P(w,d) = \sum_c P(c)P(d | c)P(w | c) = P(d) \sum_c P(c | d)P(w | c)$$

The first formulation is the symmetric formulation, where w and d are both generated from the latent class c in similar ways (using the conditional probabilities $P(d | c)$ and $P(w | c)$), whereas the second formulation is the asymmetric formulation, where, for each document d, a latent class is chosen conditionally to the document according to $P(c | d)$, and a word is then generated from that class according to $P(w | c)$.

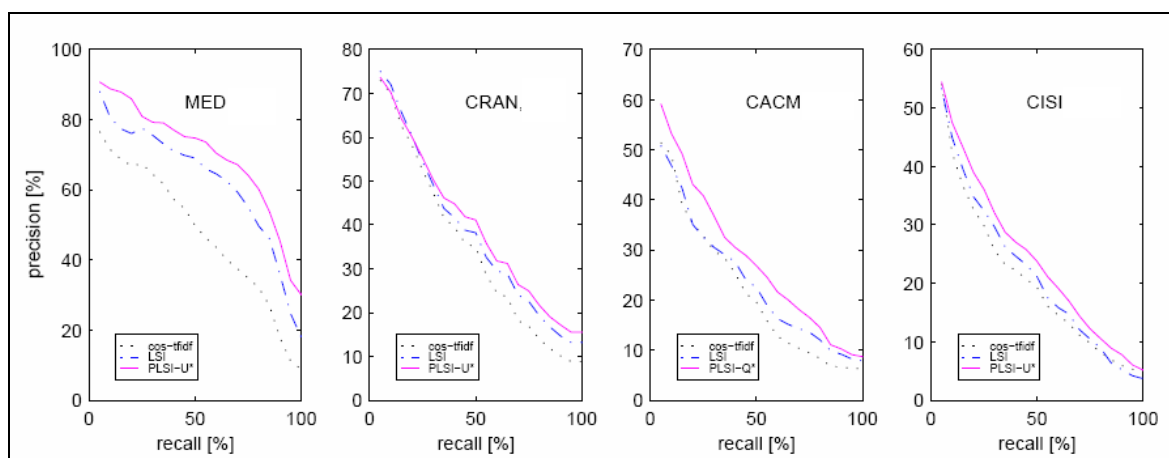


Figure 2.6 Precision-recall curves for the 4 test collections with term weighting by PLSI compare to other methods[27].

Although there are words and documents in PLSI experiments, the co-occurrence of any couple of discrete variables may be modeled in exactly the same way.

It is reported that the aspect model used in the probabilistic latent semantic analysis has severe over-fitting problems. The number of parameters grows linearly with the number of documents. In addition, although PLSI is a generative model of the documents in the collection it is estimated on, it is not a generative model of new documents [27].

PLSI was proved to be the best mathematical method in such systems compared to LSI and vector space (tf.df) as shown in Figure 2.6 which shows precision and recall for traditional centralized search system in different topics including medical and computer science[27].

2.4 Summary & Conclusion

In this chapter, we have presented existing Algorithms and architectures that are related to IR systems.

We have analyzed existed programming models such as traditional programming in addition to OOP & AOP. And programming approaches like UML & AUML with its latest progresses and developments. We found that our system should be programmed and developed using agent oriented programming approaches according to its features which are not found in OOP. But there is no AOP language, so, we tried to build our agent using JAVA language with AUML architecture.

Also, we have analyzed existed arithmetic IR Algorithms, which is related to our agent system, like Vector space algorithm, LSI and Probabilistic models. And justify why we have chosen probabilistic model in our agent which have been proved in some researches. And we will show that clearly in Chapter 4.

Chapter 3

Design of Agent architecture.

3.1 Introduction

Due to advances in technology, diverse and voluminous information is becoming available to decision makers. This presents the potential for improved decision support, but poses challenges in terms of building tools to support users in accessing, filtering, evaluating and fusing information from heterogeneous information sources.

Most reported research on Intelligent Information Agents to date has dealt with a user interacting with a single agent that has general knowledge and is capable of performing a variety of user delegated information finding tasks[19].

For each information query, the agent is responsible for accessing different information sources and integrating the results. It is believed that, given the current computational state of art, a centralized agent approach has many limitations[19]:

- (1) A single general agent would need an enormous amount of knowledge to be able to deal effectively with user information requests that cover a variety of tasks.
- (2) A centralized information agent constitutes a processing bottleneck and a “single point of failure”.
- (3) Unless the agent has beyond the state of the art learning capabilities, it would need considerable reprogramming to deal with the appearance of new agents and information sources in the environment.
- (4) Because of the complexity of the information finding and filtering task, and the large amount of information, the required processing would overwhelm a single agent.

Because of these reasons and because of the characteristics of the Internet environment, we employ a distributed collaborative collection of agents for information gathering.

We are currently working on a system where each user is associated with a set of agents which have access to the internet and select topics and keep track of the current state of the links, query, environment and user information needs.

Based on this knowledge, the agents decide what information is needed and initiate collaborative searches with other agents to get the information. During search, the agents communicate with each other to request or provide information, find information sources, filter or integrate information, and negotiate to resolve conflicts in information and task models.

The returned information is communicated to display agent or agents that possibly combine it with information from other sources (e.g. the user) and/or filter it for appropriate display to the user.

This Chapter focuses on the design of such agent for the task environment of special topic, and on the key issues that we will be addressing.

3.2 Agent Architecture

In a distributed agent framework, we conceptualize a dynamic community of agents, where multiple agents contribute services to the community. When external services or information are required by a given agent, instead of calling a known subroutine or asking a specific agent to perform a task, the agent submits a high-level expression describing the needs and attributes of the request to a specialized Facilitator agent. The Facilitator agent will make decisions about which agents are available and capable of handling sub-parts of the request, and will manage all agent interactions required to handle the complex query.

The advantage of such distributed agent architecture allows the construction of systems that are more flexible and adaptable than distributed object frameworks. Individual agents can be dynamically added to the community, extending the functionality that the agent community can provide as a whole. The agent - system is also able to adapt to available resources in a way that hard-coded distributed objects systems can't.

Using AUML (Agent UML) we will capture the MAS complexity by role decomposition and controls MAS environment dynamicity by role/agent entities separation. In terms of modeling, AUML supports the idea of UML extension toward Agent UML, which results to the integration of agent classes, role classes and interaction protocols to UML[10],[2].

3.2.1 Role A (Document Fetcher)

This Agent Role uses "wget" utility for document downloading from the internet. The link of this document is taken from a storage volume which contains a queue of links to be fetched. Links queue starts from a set of start Links presented by the administrator.

Every link from this queue is assigned estimation of usefulness of this link for seeking of new relevant documents. At the first step the newly included to this queue link is assigned number 1 as its usefulness.

The next stage of this role is the stemming stage, which leads to a logical view of documents from full text to a set of indexed terms. This stage includes Accent-spacing, noun-grouping, Stop-words-removing ... until reaching index terms from a full text.

After that, the index terms of the fetched document will be handed to Agent role C, which is responsible to figure out if the document is relevant or not.

If the document is relevant, Agent role A starts to extract all links from this document because the probability of relevance of these links is high. These links is handed directly to Agent Role D.

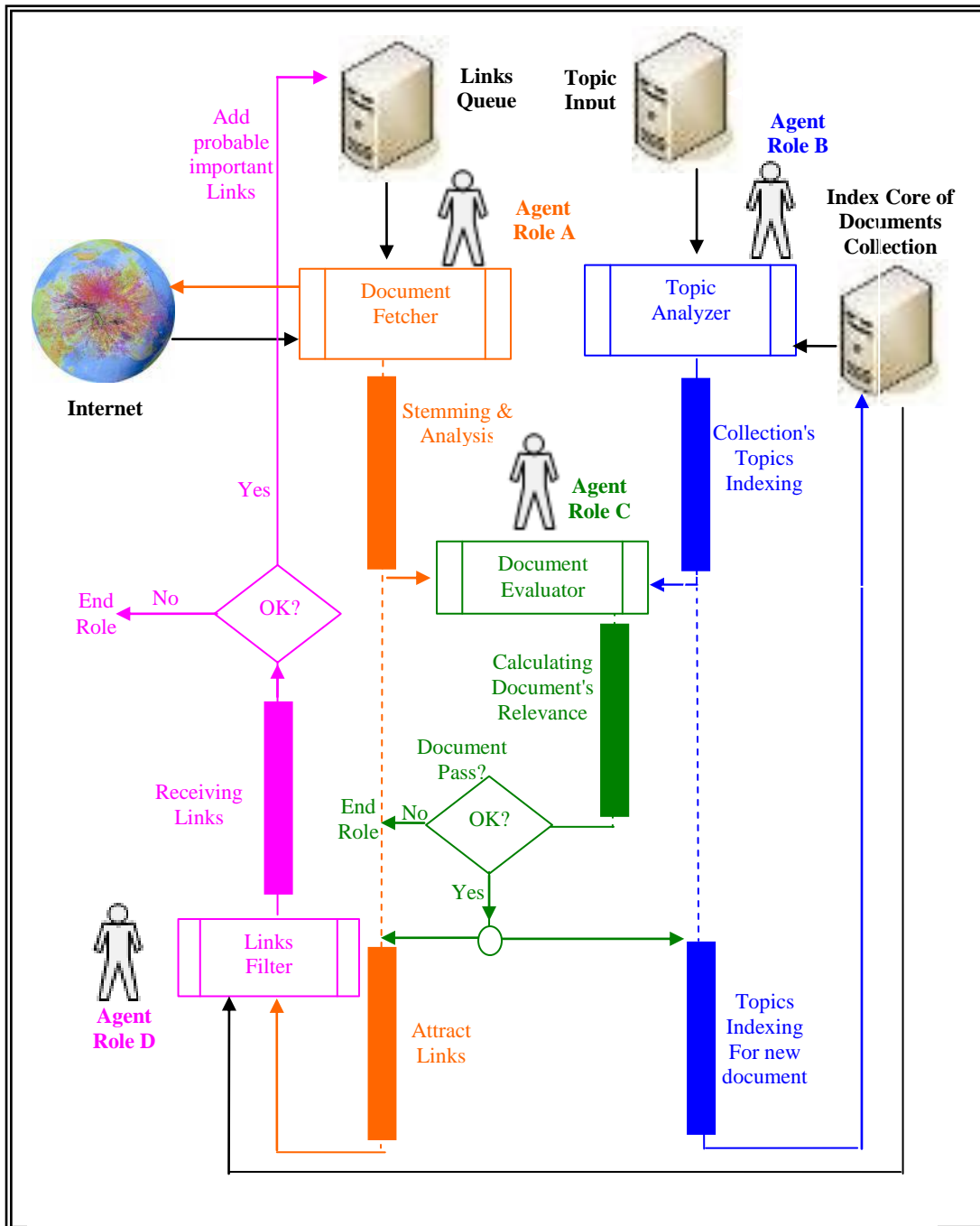


Figure 3.1 Agent Architecture AUML

3.2.2 Role B (Topic Analyzer)

Using PLSI arithmetic method, this Role is responsible for two major stages:-

Stage 1 includes receiving main target topic terms that is produced by the Administrator and stored in the topic input volume. PLSI method is used to give a weight these terms, in addition to index topic's terms that come from relevant documents stored in the Index core documents collection. Target topics terms weights are continuously modified when each new relevant document is added to the collection, at the same time these modifications are saved in Topic Input storage volume, and this loop increases the smartness of the agent. These modifications are also handed to Agent Role C, which is responsible to figure out if the document is relevant or not.

Stage 2, starts when Agent Role C decides that the fetched document is relevant, it starts to analyze the topics of document index terms using PLSI and the global frequency of topic index terms before adding the relevant document to the collection.

3.2.3 Role C (Document Evaluator)

This role is responsible to receive index terms from Role A, and Topic index from Role B, and starts to calculate the relevance of the document, and the weight of it. And then determine using special filter and comparing with a target threshold, whether the document is relevant or not. If the result is positive, then both of roles A and B start their mission on this new gift. But for negative result Role C ends its job on this not useful document.

3.2.4 Role D (Links Filter)

At every next step Role A chose from queue a link with maximum value of estimation of its usefulness, downloads it and evaluates it. If this document is accepted by evaluator then at next steps agent randomly chose links presented in its text and includes them into Links queue with usefulness estimation equal 1. If a downloaded document isn't accepted by evaluator, then estimation of usefulness of a link of a document, where link to this document occurs, is decreased. As a result, estimation of the Link usefulness is approximation of probability of relevance of a link from the document to the collection topic.

This role has to make sure that all attracted links are useful and not repeated (already checked before) in order to increase performance. And this is done with help of the stored documents collection description. So every link has to be filtered and the role decides whether to add it to Queue or not (which means to end role).

3.3 Summary & Conclusion

In this chapter we introduced the design of the agent's architecture. Using AUML including its main four major roles which are, Role A (Document Fetcher), Role B (Topic Analyzer), Role C (Document Evaluator), and Role D (Links Filter).

And we have discussed the major job of each role and how it works according to its position in the system.

We noticed the importance of the integrated job of all of the roles to perform the objectives of the system. AUML was very helpful in describing the flow of data and task handling between every role of the agent.

Chapter 4

Development Agent Algorithm.

4.1 Introduction

With the advent of digital databases and communication networks, huge repositories of textual data have become available to a large public.

Today, it is one of the great challenges in the information sciences to develop intelligent interfaces for human machine interaction which support computer users in their quest for relevant information [27].

Although the use of elaborate ergonomic elements like computer graphics and visualization has proven to be extremely fruitful to facilitate and enhance information access, progress on the more fundamental question of machine intelligence is ultimately necessary to ensure substantial progress on this issue.

In order for computers to interact more naturally with humans, one has to deal with the potential ambivalence, impreciseness, or even vagueness of user requests, and has to recognize the deference between what a user might say or do and what she or he actually meant or intended [27].

One typical scenario of human machine interaction in information retrieval is by natural language queries: the user formulates a request, e.g., by providing a number of keywords or some free-form text, and expects the system to return the relevant data in some amenable representation, e.g., in form of a ranked list of relevant documents [27].

Many retrieval methods are based on simple word matching strategies to determine the rank of relevance of a document with respect to a query.

Yet, it is well known that literal term matching has severe drawbacks, mainly due to the ambivalence of words and their unavoidable lack of precision as well as due to personal style and individual deference's in word usage [27].

Latent Semantic Indexing (LSI) is an approach to automatic indexing and information retrieval that attempts to overcome these problems by mapping documents as well as terms to a representation in the so called latent semantic space.

LSI usually takes the high dimensional vector space representation of documents based on term frequencies as a starting point and applies a dimension reducing linear projection.

The specie form of this mapping is determined by a given document collection and is based on a Singular Value Decomposition (SVD) of the corresponding term/document matrix.

The general claim is that similarities between documents or between documents and queries can be more reliably estimated in the reduced latent space representation than in the original representation.

The rationale is that documents which share frequently co-occurring terms will have a similar representation in the latent space, even if they have no terms in common.

LSI thus performs some sort of noise reduction and has the potential benefit to detect synonyms as well as words that refer to the same topic.

In many applications this has proven to result in more robust word processing.

Although LSI has been applied with remarkable success in different domains including automatic indexing (Latent Semantic Indexing, LSI), it has a number of deficits, mainly due to its unsatisfactory statistical foundation[27].

The primary goal of this chapter is to present a novel approach to LSI and factor analysis called Probabilistic Latent Semantic Analysis (PLSI), that has a solid statistical foundation, since it is based on the likelihood principle a proper generative model of the data [27].

This implies in particular that standard techniques from statistics can be applied for questions like modulating, model combination, and complexity control. In addition, the factor representation obtained by PLSI allows to deal with polysemous words and to explicitly distinguish between different meanings and different types of word usage.

4.2 PLSI Method:-

Using PLSI arithmetic method, Roles B and C is responsible to analyze the whole set of documents from this collection and create the collection description which reflects the main subjects presented in this collection. We've used for this propose probabilistic latent semantic indexing [3],[5].

The goal of the latent semantic indexing is extraction of latent factors which reflect a set of narrow topics presented in the given collection.

Let $\mathbf{z} \in \mathbf{Z} = \{\mathbf{z}_1, \dots, \mathbf{z}_k\}$ be set of these factors.

Let denote

- $P(\mathbf{z}_i)$ – probability that randomly selected document from the collection best of all corresponds to the topic \mathbf{z}_i
- $P(\mathbf{d}|\mathbf{z})$ – probability that for the given factor \mathbf{z}_i this factor best of all corresponds to the document \mathbf{d}_i
- $P(\mathbf{t}|\mathbf{z})$ – probability that for the given factor \mathbf{z}_i this factor best of all corresponds to the word \mathbf{t}_j .

Here $\mathbf{d} \in \mathbf{D} = \{\mathbf{d}_1, \dots, \mathbf{d}_N\}$ is set of all documents from the collection and $\mathbf{t} \in \mathbf{T} = \{\mathbf{t}_1, \dots, \mathbf{t}_M\}$ is set of all terms from this collection.

Functions $P(\mathbf{z}_i)$, $P(\mathbf{d}|\mathbf{z})$ and $P(\mathbf{t}|\mathbf{z})$ can be estimated in the process of a likelihood function maximization. This function is presented in the following form

$$L = \sum_d \sum_t f(d, t) \log(P(d, t)), .$$

Standard Expectation Maximization algorithm is used for maximization of this function. Two steps are executed on every iteration of this algorithm. The first one is Estimation

$$P(z | d, t) = \frac{P(z)P(d | z)P(t | z)}{\sum_{z'} P(z')P(d | z')P(t | z')}.$$

The second one is Maximization

$$P(t | z) = \frac{\sum_d n(d, t)P(z | d, t)}{\sum_{d, t'} n(d, t')P(z | d, t')}$$

$$P(d | z) = \frac{\sum_t n(d, t)P(z | d, t)}{\sum_{d', t} n(d', t)P(z | d', t)}$$

$$P(z) = \frac{1}{R} \sum_{d, t} n(d, t)P(z | d, t) , R \equiv \sum_{d, t} n(d, t)$$

To generate the collection filter we've selected the heaviest terms from T. Weight of the term t is calculated as

$$\text{weight}(t) = \sum_{z \in Z} P(z)P(t | z)$$

In our system we have used the same PLSI method to calculate the weight of every word in the (core index terms) w_{ci} which is considered to be a dictionary for the topic filter. On the other hand, we have developed a formula which is considered to be an additional formula to calculate the weight of every document W_d inside the collection, which is:-

$$W_d = \sum_t f_{t,d} \cdot w_{ci}^t$$

This filter describes the Agent's associated Collection topic and is used to quickly calculate an approximate relevance score for retrieved documents. The goal of the W_d filter is to extract relevant pages, and abandon junk pages, recommended to the Collection, where the final arbitrator of relevancy is the collection itself .

Also, this filter monitors a stream of incoming documents and selects those that match a certain query. The initial topic filter is set by the Collection, W_d basically consists of a term vector and a recommendation threshold τ . The term vector contains terms t with associated term weights w_{ci} . The threshold τ is a positive number used to decide whether a document is judged relevant enough to be recommended to the Collection .

A document profile as well as a filter name is delivered to the Document Evaluation role.

If $f_{t,d}$ is the frequency of term t in document d , then the document weight W_d is calculated by the Document Evaluation Role using mentioned formula.

Only if $W_d > \tau$, then the document d is recommended to the Collection .

This Filter is responsible for the following:

- Recommend document.
- Abandon document.
- Rebuild core terms.
- Automatic refinement a filter on the basis of accurate W_d from the Collection to improve the quick initial evaluation made by the document evaluation role.
- Rank the relevant documents output.

To better reflect the Collection's information needs, the Agent can automatically refine its filter based on relevance feedback from the Collection. Since the Collection feedback arrives continuously, the topic filter needs to be iteratively refined .

When any document succeeds in passing throw the filter and threshold the system is supposed to use this document to modify and re-index the core terms according to the global frequency (gf) of each term which is calculated too.

We mean by global frequency (gf) by the number (count) of repetition of a term among succeeded documents and not inside document (local frequency – lf). The increase of (gf) for any term should increase the weight of this term. A suggested formula could be helpful in rebuilding the core index terms of the agent.

If a term shows up in a new relevant document weight will be:

$$w'(t) = w(t) + 1/N_D$$

Or

$$w'(t) = w(t) - 1/N_D$$

$w(t)$: current weight of the term.

$w'(t)$: new weight for the term.

N_D : number of all accepted documents.

The range of the weight is in $[0..1]$, but this formula is not implemented yet in this work. It is one of our future works. We recommend this part to be studied carefully.

4.3 How it works?

The goal of using PLSI method is to analyze the whole set of Administrator's queries which reflects information need of him.

This analysis can be used to find new subjects which are interested to him but poorly presented in the collection core index.

In order to do so we've used the following approach. At first graph of all words used in the Administrator's terms was created. Every word was presented as a vertex of this graph. Two vertices are joined with an edge if and only if the pair of corresponded words occurs in the same query.

Every vertex should have a weight which reflects the role of this word in the collection subject. Some of these words are presented in the collection core and we can use probabilistic latent semantic indexing to calculate their weights. But a part of words presented in the queries can be new (not presented in the collection core). To estimate their weights we've used the following method.

We suppose that weight of every new word should be equal to the average value of weights of words which are neighbors of this word. We've used iteration algorithm to estimate weights of all new words according to this proposal. All information about queries words and their weights is stored as queries statistics.

4.4 Summary & Conclusion

In this chapter we have discussed in details the development of the agent algorithm which depends on PLSI methods and how it works in our agent. We have presented the mathematical functions that we used inside the agent.

Calculating weights for documents and terms in both document and topics using PLSI method and threshold formula, was very helpful in estimating the relevance of each document to agent's object topic.

PLSI was proved to be the best mathematical method in such systems compared to LSI and vector space (tf.df) as shown in Figure 2.6 which shows precision and recall for traditional centralized search system in different topics including medical and computer science[27].

Chapter 5

Implementation and testing.

5.1 Introduction

Intelligent Agents are currently the subject of research by a wide and varied community worldwide. Intelligent agents have received various, if not contradictory, definitions; this is not surprising, given the wide variety of goals set by different researchers.

In general, researchers agree that an agent is a complex object that shows some degree of autonomy and social ability, and combines pro-active and reactive behaviors[16].

To help put agents into a correct engineering perspective, we have included some general considerations regarding what has been called 'agent oriented programming' AOP [12].

Broadly speaking, our agent can be seen as a process that pursues a number of goals over a long period of time (relative to the application domain), and somehow reacts and adapts to the evolution of its environment. A multi-agent system attempts to pursue some kind of common goals by a combination of cooperation, negotiation and competition among agents.

From an engineering perspective, our agent-based systems differ from traditional distributed systems because of their emphasis on distributed problem solving; programming is at a higher level of abstraction than is currently allowed by mainstream languages and methodologies.

Distributed object oriented applications are commonly developed by creating or customizing classes at different levels of abstraction and stacking them, starting from some communications infrastructure at the lowest layer.

Typically, a traditional system does not incorporate any representation of 2 global or per-process goals, which remain in the minds of its designers and are somehow lost in the process of top-down decomposition and distribution over the network.

In contrast, building our agent-based system commonly follows a process that is the reverse of what is described above. Our agent is described in terms of its high-level objectives, which usually consist of handling certain messages and events and achieving given goals; multi-agent frameworks may allow the declaration of objectives for the whole system.

At runtime, it is possible to trace the reasons (that is, the high-level objectives) that triggered the observed behaviors of an agent.

Furthermore, agents can often choose between different courses of actions (that is, scripts, rules, plans, and so on) in order to pursue their objectives, and can try many of them sequentially or concurrently, depending on their state and that of the environment[16].

In many instances, agent development frameworks are based on, or allow access to, other AI technologies (for example, logic or functional programming, knowledge bases, fuzzy logic, and so on) [16].

Our agent-based system could be seen as little more than an application of patterns such as the Active Object. However, its development process and tools are different from conventional distributed programming.

These tools enable the declaration of the objectives and behavior of agents at a higher level of abstraction and support a corresponding view of the activity of the system at run-time.

Among other advantages, this allows a richer set of distributed architectures than client-server (including cooperation in teams, market-style negotiation for the distribution of tasks among participants, and so on) and rapid application development.

The implementation of distributed procedures tends to be direct and straightforward. Thus a framework for intelligent agents is more than just a scripting language or a set of components for distributed applications.

Such a framework must facilitate correspondence of the observed behavior of our agent to some high-level objectives. The framework must also take care of tasks being pursued concurrently and prioritize them when required.

Importantly, it must help in coordinating potentially conflicting tasks, in choosing the best course of action when alternatives are possible and reacting appropriately on failure.

Managing these aspects is sometimes referred to as ‘meta-programming’, they are first-order elements of agent programming and represent another important distinction compared with traditional procedural or object-oriented programming.

Most frameworks currently available in the research environment have shortcomings. For example, some of them are based on languages or technologies considered (quite rightly) esoteric by mainstream engineering.

Also, a very high level language or framework is usually not appropriate to solve problems for which proven, efficient algorithmic solutions are available.

Moreover, agent-based applications require access to existing computing infrastructures and software in order to re-use components or information already in place and to add new functionality to legacy systems (by either ‘wrapping’ them into an agent infrastructure or adding high-level procedures, such as business rules, as an external component).

These considerations are some of the motivations for JAVA, which we have chosen to program our agent's classes.

5.2 Agent Major Classes

Programming such agent is not like programming similar software. We tried to use Java as a programming language to perform the task and the objectives of our agent. We went through a lot of difficulties in building agent's classes. So, we started from Class "AgentA" as shown in Appendix Figure 7.1.

In this class, we tried to construct a simple interface frame that will be used as an agent-user interface. Its main object is to let the user initiates the agent and enter his options. This interface is not complicated and it appears like a small window in the middle of the screen. As shown in Figure 5.1.

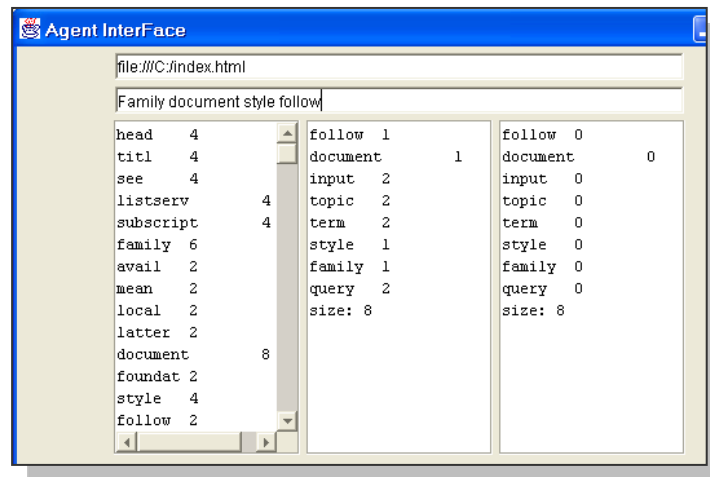


Figure 5.1 Agent-User Interface

This Class is surely created by Main method as well as every Java class. Refer to Appendix Figure 7.2, shows the call for initiating "AgentA" class.

In Java, all related classes should be compiled and loaded to main memory in order to run the interface frame successfully. In Figure 5.2, we can see the agent's interface running with related classes to perform the experiments of this thesis.

Chapter 5: Implementation and testing.

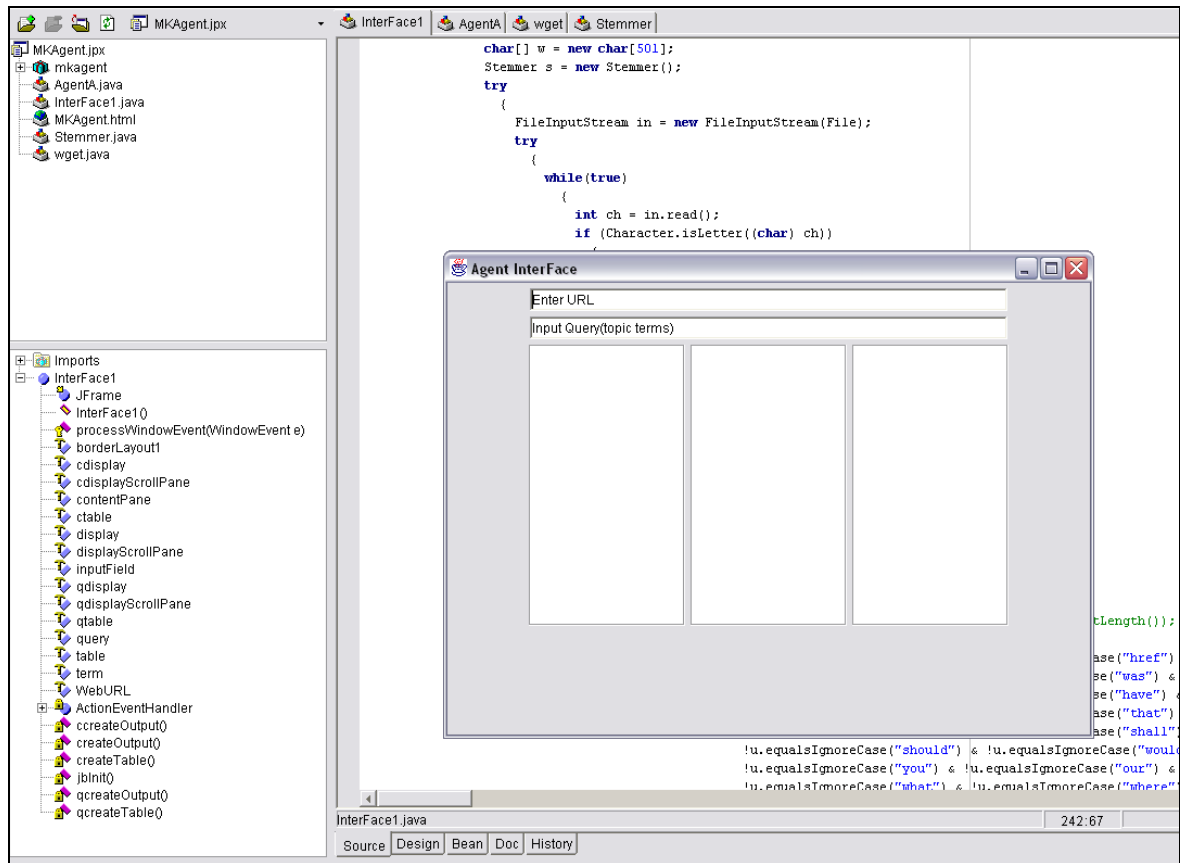


Figure 5.2 Agent's Interface with related classes

In the next section we will discuss the main classes of our system that is critical to perform our experiments.

5.2.1 Main Class.

In our system, the main class is called "Interface1", which contains the main procedure of the agent, and a reference intelligent point for the system. In Figure 5.3, we show the description of the class and its contents of sub-classes and parameters, variables, function, and declarations.

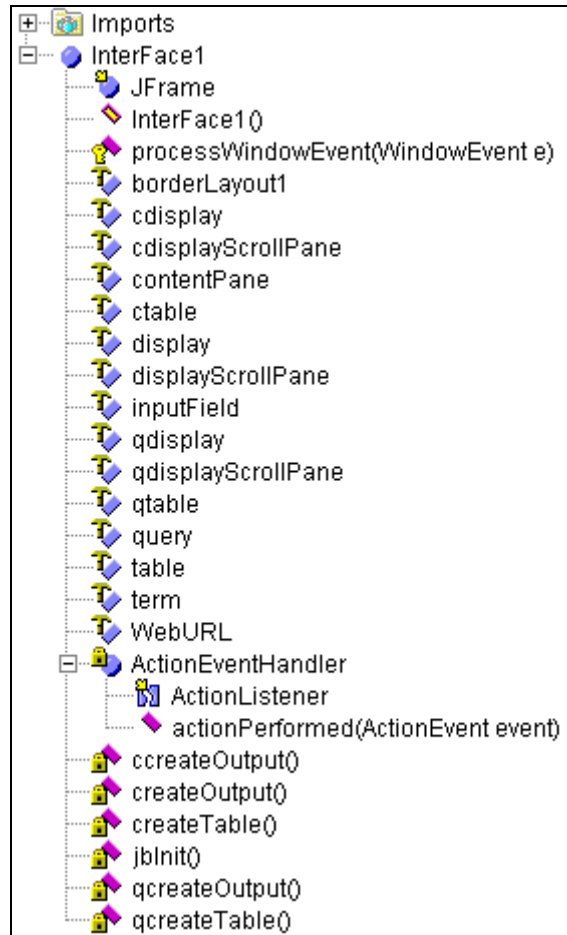


Figure 5.3 InterFace1 Class

We will try to describe in details some of these properties of Interface1 class. As we show, this class is created after it has been called from AgentA class which contains the main method (Figure 5.4).

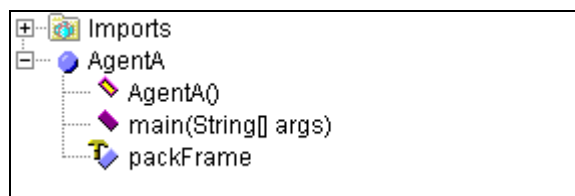


Figure 5.4 AgentA Class

This class starts with constructing the main frame of the interface window that will appear after AgentA is initiated. It contains the text-fields, text-areas, scroll panels, hash tables, and some other parameters that is essential for the agent to start building his own peripherals which he need to perform user's instructions and queries, as shown in Appendix Figure 7.3.

All important components like web address, topic terms, term's frequencies, and files are declared in this class. An important method in Interface1 class is "createTable()".

This method (Appendix Figure 7.4) is responsible for creating a table of term frequencies contained in the input document (brought from internet). The same steps are done to create "qtable" from user's terms in the input-field (Appendix Figure 7.5).

Another important method of interface1 is "createOutput()" which is essential in performing an output in the window showing table's values (terms and their frequencies) as shown in Appendix Figure 7.6.

The same steps of this method is used in "qcreatOutput()" which is used in performing an output in the window showing query's values (terms and their frequencies) as shown in Appendix Figure 7.7.

We used the same steps in creating an output for the shared terms between table and query. This job was handled by "ccreatOutput()" method. In other words, it shows the terms of the query that exists in table and their frequencies in table. See Appendix Figure 7.8.

As well as all Java applications there should be a method considered to be "component initiator". And this Job is handled in our thesis by method called "jbInit()". As shown in Appendix Figure 7.9.

The most important part of Interface1 class is that to bring internet document from a user-given website address or from an addresses queue.

An inner class in interface1 class called "ActionEventHandler" which includes an action that should be handled after an event happened inside the interface frame's components.

When a text field called "WebURL" the program initiates directly a class that is called "WGET" which its main job is to bring that document from the internet and save it into a local file inside the computer. As shown in Appendix Figure 7.10.

The brought file in local volume will be named "Website.html".

Another important part of interface1 class is to call "stemmer" class, to stem every string inside the brought file. The stemming stage of the document includes ignoring some words and phrases, before creating the table. As shown in Appendix Figure 7.11.

5.2.2 WGET Class.

An important class, which helps the agent to bring any document from the internet, and fetch any web page address. Similar to WGET command in UNIX, we presented the WGET class. It contains two important method, "creatAFile" and "get" as shown in Figure 5.5.

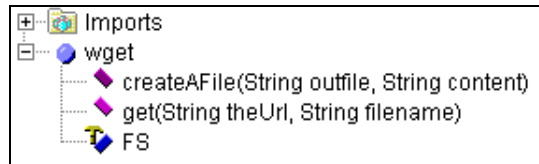


Figure 5.5 WGET Class

This class is essential to the system as independent class to be initiated and called when it is needed. It also, saves the retrieved file into a local storage volume. See Appendix Figure 7.12.

The "wget" class uses "createAFile" method in order to create a local file contains the contents of the brought website. As shown in Appendix Figure 7.13.

5.2.3 Stemming Class.

Every loaded document should be put under an essential treatment. All words inside this document should be stemmed in order to increase the efficiency of the system.

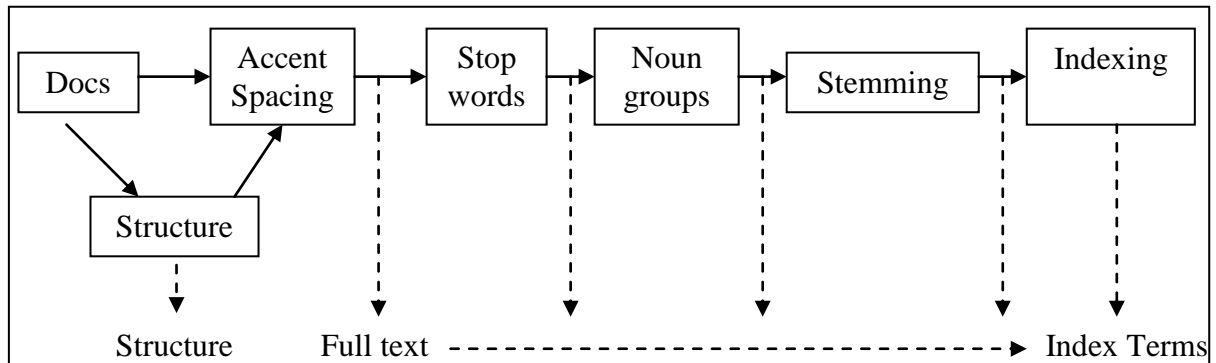


Figure 5.6 Converting full text to indexed terms through stemming

Our object is to get to a logical view of documents from full text to a set of indexed terms (Figure 5.6). We mean by "Full text (doc)" by logical view of terms (representation) because modern approach IR agents are making it possible to represent a document by its full set words.

are steps like:-

- 1- Eliminate stop words: such as articles and connectives.
- 2- Identifications of Noun Groups: verb, adverb, adjectives.
- 3- Use stemming: Identification grammatical root.
- 4- Indexing : set of terms.

The "Stemmer" class agent (Figure 5.7) is responsible to do the part of Transformation (Text Operations) in our system.

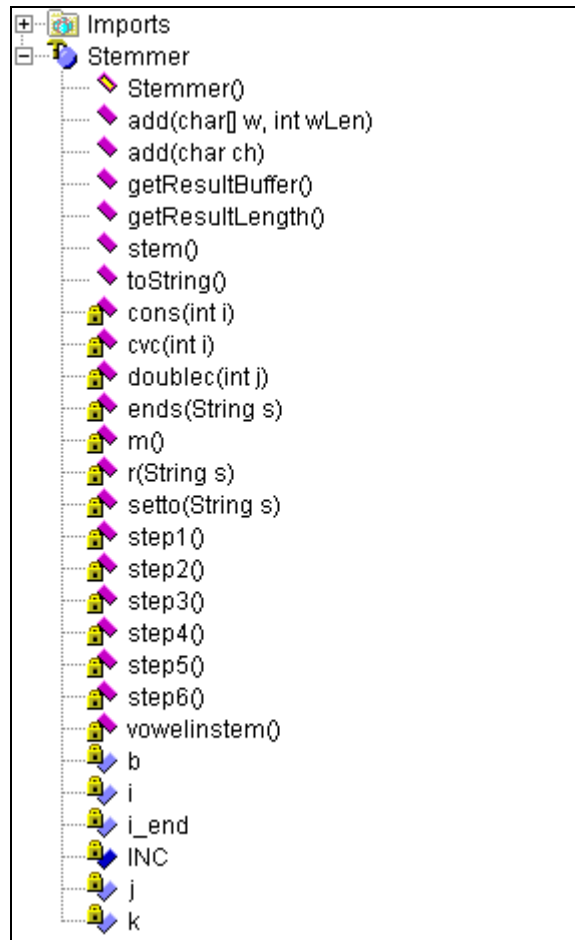


Figure 5.7 Stemmer Class

By implementing the stemmer class in order to transform a word into its root form, the input word can be provided a character at a time (by calling "add()") method -shown in Appendix Figure 7.15, or at once (by calling one of the various stem() methods).

In Appendix Figure 7.14, we can see the declaration and main parameters of stemmer class.

The same steps in add() method, is used in other case of characters but faster that the previous one as shown in Appendix Figure 7.16.

After a word has been stemmed, it can be retrieved by "toString()" method. (Appendix Figure 7.17)

Stemmer class contains the ability of returning the length of the word resulting from stemming process using i_end variable And "getResultLength()" function. (Appendix Figure 7.18)

Sometime we need to return a reference to a character buffer containing the results of the stemming process. This can be done with the help of "getResultBuffer()" method. As shown in Appendix Figure 7.19.

Appendix Figure 7.20 shows a part of stemmer that is concerned with consonant cases.

The following function method can measure the number of consonants not vowels in the word to be treated later on. See Appendix Figure 7.21.

For sure we do need a vowel checker to indicate if or not the word contains vowel. See Appendix Figure 7.22. Note that this method depends on `cons()`.

In some cases in English language we do need to indicate if there exists a double consonant in the word or not. This job is handled by "doublec() method " as shown in Appendix Figure 7.23.

Special cases in English language when some words contains consonant-vowel-consonant and the second consonant is not w,x, or y, we need the following step to restore an "e" at the end of the word. (e.g. cav(e), lov(e) ...). See Appendix Figure 7.24.

Appendix Figure 7.25 shows a string's end checker which is helpful in some cases in English words.

After a word is stemmed we need to take part of word's letters and ignore others. So we use the following method do this job. See Appendix Figure 7.26 And Appendix Figure 7.27.

The stemming procedure on any word should pass through many steps. We will discuss these steps in the next methods.

Let us start with step1, which removes plurals and –ed or –ing from the word that is to be stemmed. See Appendix Figure 7.28.

Step 2, is important to turn terminal y to i when there is another vowel in the string.(Appendix Figure 7.29).

Step 3, is important to map double suffices to single ones, for special cases. See Appendix Figure 7.30

Step 4, uses strategy similar to step 3 but deals with cases of words contains –ic-, -full, -ness. etc. See Appendix Figure 7.31.

Step 5, takes of word's ends like –ant, ence, etc. in some cases. As presented in Appendix Figure 7.32.

Last step is step 6 , which removes "e" in some word that lasts with it. See Appendix Figure 7.33

All words that are needed to be stemmed should go through the six steps. The method called "stem()" applies all these steps in every word. As shown in Appendix Figure 7.34.

5.2.4 PLSI Class.

In this class, we tried to implement PLSI mathematical method inside our agent. Its main job is to evaluate (estimate) the weights of documents, topics and terms. We have used three major C++ language programs to calculate our readings.

Let us start with "mex_EMstep.c"¹ (Appendix Figure 7.35). mex_EMstep performs one step of (T)EM given the parameters

usage: $Y = \text{mex_EMstep}(X, C, Pw_z, Pz_d)$
or $Y = \text{mex_EMstep}(X, C, Pw_z, Pz_d, \text{beta})$

where 'X' is the term-document matrix, 'C' the normalization constant (evaluated at the non zero points of 'X', 'Pw_z' the conditional distribution over words given the topics, 'Pz_d' the document conditioned distribution over the topics. 'beta' \elem (0,1) for tempered EM. (default: 1) 'X' and 'C' have to be sparse (and of the same structure).

Another important C++ program is "mex_Pw_d.c", which computes the normalization constant during learning for PLSI. The elements are computed only at those positions needed. See Appendix Figure 7.36.

The third C++ program used in PLSI implementation was "mex_logL.c". Which syntax is

$\text{logL} = \text{mex_logL}(X, Pw_d, Pd)$

where X is the term-document matrix, Pw_d the distribution over the words given the documents and Pd the prior distribution over the documents. X and Pw_d need to be sparse (and of the same structure). See Appendix Figure 7.37.

¹ Peter Gehler, Max Planck Institute for biological Cybernetics, pgehler@tuebingen.mpg.de, Feb 2006

5.2.5 Get Links Class.

A very important part in the program, is "Getlinks" class. As mentioned before in chapter three, if a document passes the threshold condition and found relevance to agent's object topic, the links inside this document has a good possibility to be relevant too. The agent should be able to get these links to be fetched in the queue after have been checked by "Role D". Appendix Figure 7.38, shows this part of our system.

This job is done with the help of "getReader()" methods, which checks all parts of the document begins with "http:" in order to recognize a link inside a document. See Appendix Figure 7.39.

5.3 Testing Experiments

In order to test our system, -using the previously mentioned methods and classes- we have made a collection of documents about computer science topic. Some of these documents are relevant to our topic which we have selected to be a test-topic for our system.

The documents we have collected, contains terms and links to each other. The following table 5.1 shows the relevance of each document in our test-collection. Then we applied two tests to this collection.

The first test was by applying traditional topic-search system that uses PLSI method on this collection. In fact we used a duplicate of our system but has no filters or thresholds in it. Then we calculated precision and recall. The details will be presented later in the next section.

The second test was by applying our topic-search agent system that uses PLSI method on this collection with our filter with thresholds inside it. Then we calculated precision and recall too. The details will be presented later in the next sections.

Comparing the results of both experiments indicate the precision and recall of our architecture and design is better than the other one. In chapter six, we will discuss in details the advantages of our system.

As we can see in table 5.1, more than a hundred documents are involved in the collection and used in both experiments one and two. The relevant documents are selected carefully to be about a topic that we are familiar with. for example, the first experiment was about computer science which we can decide whether the document is relevant or not.

Table 5.1 Relevance of Test Document Collection

Docs	Relevant?	Docs	Relevant?	Docs	Relevant?	Docs	Relevant?
D001	FALSE	D033	FALSE	D065	FALSE	D097	TRUE
D002	FALSE	D034	TRUE	D066	TRUE	D098	TRUE
D003	FALSE	D035	TRUE	D067	FALSE	D099	FALSE
D004	FALSE	D036	TRUE	D068	TRUE	D100	FALSE
D005	TRUE	D037	FALSE	D069	FALSE	D101	FALSE
D006	FALSE	D038	FALSE	D070	TRUE	D102	TRUE
D007	FALSE	D039	FALSE	D071	FALSE	D103	TRUE
D008	FALSE	D040	FALSE	D072	TRUE	D104	FALSE
D009	FALSE	D041	FALSE	D073	FALSE	D105	FALSE
D010	FALSE	D042	FALSE	D074	FALSE	D106	TRUE
D011	FALSE	D043	FALSE	D075	FALSE	D107	FALSE
D012	FALSE	D044	FALSE	D076	FALSE	D108	FALSE
D013	FALSE	D045	TRUE	D077	FALSE	D109	TRUE
D014	FALSE	D046	FALSE	D078	FALSE	D110	FALSE
D015	FALSE	D047	FALSE	D079	FALSE	D111	FALSE
D016	FALSE	D048	FALSE	D080	FALSE	D112	FALSE
D017	FALSE	D049	FALSE	D081	TRUE	D113	TRUE
D018	TRUE	D050	FALSE	D082	FALSE	D114	FALSE
D019	TRUE	D051	FALSE	D083	FALSE	D115	FALSE
D020	TRUE	D052	FALSE	D084	FALSE	D116	FALSE
D021	FALSE	D053	FALSE	D085	FALSE	D117	FALSE
D022	FALSE	D054	FALSE	D086	FALSE	D118	FALSE
D023	FALSE	D055	FALSE	D087	FALSE	D119	FALSE
D024	FALSE	D056	FALSE	D088	FALSE	D120	FALSE
D025	FALSE	D057	TRUE	D089	FALSE	D121	FALSE
D026	FALSE	D058	FALSE	D090	TRUE	D122	TRUE
D027	FALSE	D059	FALSE	D091	FALSE	D123	FALSE
D028	TRUE	D060	FALSE	D092	FALSE	D124	TRUE
D029	FALSE	D061	FALSE	D093	TRUE	D125	TRUE
D030	FALSE	D062	FALSE	D094	FALSE	D126	FALSE
D031	FALSE	D063	FALSE	D095	FALSE	D127	TRUE
D032	FALSE	D064	FALSE	D096	TRUE	D128	TRUE

The next step was to select terms in our selected topic and apply our software to get the document-term matrix (frequencies). Then to calculate the weight of each term in the topic using methods mentioned in chapter 4. using the following formula:

$$\text{weight}(t) = \sum_{z \in Z} P(z)P(t | z)$$

The following four tables, Table 5.2, 5.3, 5.4 and 5.5 shows the document-term matrix with term's weights in the selected topic

Table 5.2 Document[1..32]-Term matrix with P(w/z)

Weight	0.7258	0.98858	0.40884	0.07882	0.55437	0.18809	0.84872	0.71448	0.12924	0.95807	0.86022	0.6425
Docs1	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11	t12
D1	93	85	95	14	59	66	50	28	91	73	47	96
D2	58	47	65	32	93	74	96	75	81	16	87	21
D3	31	53	9	72	36	3	23	38	57	68	7	3
D4	77	84	43	68	70	17	46	3	50	4	99	40
D5	76	24	43	30	20	12	84	8	54	89	57	0
D6	54	68	15	51	6	57	16	20	23	35	4	84
D7	30	60	29	60	8	72	27	5	68	81	17	26
D8	96	5	83	74	25	21	16	37	41	38	42	17
D9	39	8	97	21	27	64	94	95	60	61	25	18
D10	1	0	89	58	40	7	43	56	59	0	3	85
D11	92	0	12	46	58	10	32	29	22	58	17	35
D12	56	27	39	29	91	78	60	66	42	97	61	90
D13	67	76	12	71	90	87	9	8	97	73	56	89
D14	29	91	97	6	30	7	97	73	37	19	52	99
D15	54	6	3	42	36	56	32	57	47	5	88	51
D16	56	12	69	76	45	70	37	52	34	41	45	17
D17	42	83	0	84	58	55	11	8	64	65	62	0
D18	36	97	55	13	31	71	19	19	97	93	64	27
D19	88	27	1	1	57	15	95	88	82	52	97	57
D20	43	5	49	21	19	19	75	16	64	15	71	20
D21	5	28	68	20	52	34	36	30	45	98	38	3
D22	59	44	2	11	45	63	30	62	5	51	46	3
D23	3	0	75	80	67	0	97	80	92	51	69	72
D24	61	34	45	61	95	60	54	61	20	95	60	31
D25	50	5	66	71	32	52	64	12	40	31	41	2
D26	42	81	72	76	83	31	73	54	18	62	93	74
D27	9	6	71	20	76	14	31	71	18	4	7	24
D28	51	89	0	47	2	2	99	17	27	42	2	13
D29	56	56	79	50	72	15	70	81	4	84	8	1
D30	17	42	21	64	65	49	99	58	16	65	58	70
D31	84	70	26	4	33	43	11	87	7	3	68	79
D32	33	75	69	79	95	15	35	84	14	87	58	89

Table 5.3 Document[33..64]-Term matrix with P(w/z)

Weight	0.7258	0.98858	0.40884	0.07882	0.55437	0.18809	0.84872	0.71448	0.12924	0.95807	0.86022	0.6425
	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11	t12
D33	86	55	75	89	96	80	49	45	3	58	15	9
D34	32	66	13	25	9	79	58	10	19	88	87	95
D35	80	81	65	8	0	5	32	62	81	67	7	93
D36	93	92	41	41	6	24	65	34	7	63	75	73
D37	9	80	96	0	45	96	95	84	72	42	37	29
D38	76	50	6	82	48	27	9	16	59	26	52	68
D39	20	5	33	46	10	87	69	16	78	98	75	38
D40	62	68	90	65	97	7	88	77	57	41	10	43
D41	94	74	32	27	98	22	91	79	29	29	17	3
D42	21	67	49	16	68	88	72	82	67	24	53	59
D43	87	64	0	88	67	27	84	79	90	23	3	95
D44	57	43	54	53	45	88	35	14	89	59	10	57
D45	2	66	7	11	42	52	20	39	34	73	95	37
D46	45	54	76	40	82	17	14	21	59	10	73	77
D47	68	66	96	32	50	40	75	92	16	91	43	2
D48	59	63	91	90	81	96	58	33	40	50	39	34
D49	62	80	81	88	87	32	39	98	92	16	12	56
D50	26	29	21	86	76	10	67	45	44	32	22	6
D51	34	38	27	49	44	5	60	99	34	49	91	9
D52	45	52	33	65	38	9	37	41	4	22	84	67
D53	51	10	12	68	77	17	71	35	53	11	67	47
D54	14	59	54	70	67	37	22	58	78	84	22	17
D55	21	19	85	93	70	35	7	99	14	60	6	86
D56	25	97	99	93	75	76	99	88	72	70	6	54
D57	37	58	18	83	31	3	86	11	45	49	84	25
D58	43	10	58	55	54	14	13	30	8	87	24	16
D59	58	79	55	38	3	58	38	77	92	27	43	44
D60	57	88	7	19	98	65	23	21	53	7	98	40
D61	29	85	48	2	55	8	23	87	81	19	36	52
D62	89	51	67	77	96	36	83	37	56	54	56	39
D63	23	53	90	29	39	70	95	98	80	7	45	66
D64	90	26	0	93	36	39	87	91	38	8	47	6

Table 5.4 Document[65..96]-Term matrix with P(w/z)

Weight	0.7258	0.98858	0.40884	0.07882	0.55437	0.18809	0.84872	0.71448	0.12924	0.95807	0.86022	0.6425
	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11	t12
D65	0	0	0	22	0	81	0	0	0	74	0	15
D66	0	85	19	0	17	0	55	29	0	65	68	0
D67	13	61	2	65	0	0	0	0	0	0	40	97
D68	50	82	0	0	0	0	87	0	0	0	60	78
D69	80	0	48	0	0	0	47	68	50	0	0	0
D70	0	29	0	0	0	8	0	0	97	0	71	16
D71	39	13	17	66	0	87	0	0	0	0	0	0
D72	0	48	0	0	21	0	0	77	53	67	83	0
D73	0	0	50	53	0	28	85	0	13	98	11	0
D74	25	50	0	0	0	58	93	42	0	87	0	70
D75	52	0	97	0	45	1	62	73	0	23	0	51
D76	0	24	0	77	0	11	0	0	0	77	7	0
D77	0	0	35	45	0	0	91	35	70	75	0	0
D78	72	0	68	56	0	0	0	92	86	0	0	0
D79	0	0	0	31	57	24	0	49	0	0	0	0
D80	42	0	98	16	96	46	0	32	96	77	98	0
D81	0	0	0	0	0	0	76	0	0	41	0	62
D82	0	0	28	86	38	8	0	0	0	66	17	76
D83	0	21	14	80	0	23	8	0	9	0	85	0
D84	16	0	0	69	42	97	0	93	24	0	19	0
D85	0	0	78	50	38	15	0	55	0	0	0	57
D86	0	96	0	66	0	0	4	0	0	23	0	0
D87	0	23	13	9	46	0	0	93	93	65	74	93
D88	0	0	38	0	0	0	28	40	30	72	0	0
D89	0	0	0	55	0	50	71	20	0	0	4	89
D90	32	43	38	0	23	0	65	0	71	0	0	79
D91	87	21	87	1	17	0	0	86	0	4	0	0
D92	40	0	97	87	45	41	0	82	74	0	0	14
D93	0	32	0	0	68	28	50	0	0	37	65	13
D94	0	66	33	15	0	93	93	48	8	54	57	26
D95	0	23	49	53	0	0	0	0	0	68	32	0
D96	0	71	0	0	75	0	0	25	0	0	96	0

Table 5.5 Document[97..128]-Term matrix with P(w/z)

Weight	0.7258	0.98858	0.40884	0.07882	0.55437	0.18809	0.84872	0.71448	0.12924	0.95807	0.86022	0.6425
	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11	t12
D97	0	0	0	0	0	0	24	0	86	0	66	38
D98	95	0	0	0	0	0	0	0	64	0	47	0
D99	52	39	0	38	20	0	0	43	0	60	48	3
D100	27	0	39	7	79	0	13	93	0	32	45	45
D101	0	0	89	0	0	11	1	0	0	77	0	0
D102	0	63	7	26	33	0	0	38	85	48	75	47
D103	0	77	94	0	0	0	0	0	87	0	12	0
D104	53	94	0	0	0	96	0	46	95	58	0	0
D105	47	2	15	51	0	0	0	4	83	11	0	0
D106	0	0	0	0	51	0	53	29	40	87	30	0
D107	18	0	47	81	13	9	96	34	26	18	0	0
D108	0	0	0	0	0	85	57	0	95	0	0	0
D109	52	0	46	0	27	0	78	0	77	0	26	79
D110	0	0	0	0	0	61	0	0	0	70	0	0
D111	88	99	30	0	0	85	0	0	0	3	0	57
D112	16	0	16	0	0	55	0	71	19	82	32	0
D113	49	61	0	9	0	0	21	0	0	98	0	0
D114	0	0	0	0	0	56	91	0	0	0	27	62
D115	0	0	2	0	67	31	29	68	41	84	24	0
D116	56	92	7	14	69	45	0	89	87	0	70	0
D117	0	0	0	99	31	80	6	0	34	0	44	0
D118	46	86	86	0	45	0	10	63	0	0	0	21
D119	17	0	0	48	0	18	0	94	0	0	53	0
D120	0	1	21	81	0	19	0	1	98	30	78	61
D121	0	53	48	75	0	49	40	17	0	0	9	0
D122	0	89	0	0	0	0	0	19	84	0	0	0
D123	37	0	0	12	0	66	0	78	0	0	0	34
D124	53	85	74	6	0	0	13	0	45	21	0	39
D125	0	94	0	0	0	6	0	0	0	0	75	62
D126	0	0	0	72	54	19	56	0	77	14	78	0
D127	0	42	0	4	0	0	29	0	6	63	0	0
D128	0	64	44	31	0	0	0	0	28	12	20	0

5.3.1 Experiment 1

In this experiment we started by applying traditional topic-search system that uses PLSI method on this collection. In fact we used a system that is similar to our system but has no filters or thresholds in it. Then we calculated precision and recall. The details are as shown in Tables 5.10, 5.11, 5.12 and 5.13. Note that the sequence of documents to be fetched was according to the links queue and those are inside the documents starting from document 82.

Table 5.10 Experiment 1 results (Precision & Recall) WGET[1..32]

WGET	Documents	Weight	Relevant	R-R	R-N	I-R	I-N	Precision	Recall
1	D082	0.504781	FALSE	0	29	1	98	0	0
2	D045	0.682782	TRUE	1	28	1	98	0.5	0.034483
3	D035	0.692718	TRUE	2	27	1	98	0.666667	0.068966
4	D008	0.544601	FALSE	2	27	2	97	0.5	0.068966
5	D062	0.616768	FALSE	2	27	3	96	0.4	0.068966
6	D106	0.744729	TRUE	3	26	3	96	0.5	0.103448
7	D069	0.594778	FALSE	3	26	4	95	0.428571	0.103448
8	D072	0.713054	TRUE	4	25	4	95	0.5	0.137931
9	D043	0.60354	FALSE	4	25	5	94	0.444444	0.137931
10	D108	0.655617	FALSE	4	25	6	93	0.4	0.137931
11	D014	0.660878	FALSE	4	25	7	92	0.363636	0.137931
12	D031	0.595368	FALSE	4	25	8	91	0.333333	0.137931
13	D070	0.879918	TRUE	5	24	8	91	0.384615	0.172414
14	D090	0.763442	TRUE	6	23	8	91	0.428571	0.206897
15	D052	0.604035	FALSE	6	23	9	90	0.4	0.206897
16	D076	0.531237	FALSE	6	23	10	89	0.375	0.206897
17	D079	0.278816	FALSE	6	23	11	88	0.352941	0.206897
18	D015	0.571226	FALSE	6	23	12	87	0.333333	0.206897
19	D028	0.737799	TRUE	7	22	12	87	0.368421	0.241379
20	D047	0.594696	FALSE	7	22	13	86	0.35	0.241379
21	D024	0.577798	FALSE	7	22	14	85	0.333333	0.241379
22	D056	0.564595	FALSE	7	22	15	84	0.318182	0.241379
23	D054	0.589669	FALSE	7	22	16	83	0.304348	0.241379
24	D071	0.313847	FALSE	7	22	17	82	0.291667	0.241379
25	D018	0.707376	TRUE	8	21	17	82	0.32	0.275862
26	D006	0.591998	FALSE	8	21	18	81	0.307692	0.275862
27	D068	0.820505	FALSE	8	21	19	80	0.296296	0.275862
28	D012	0.60342	FALSE	8	21	20	79	0.285714	0.275862
29	D041	0.624845	FALSE	8	21	21	78	0.275862	0.275862
30	D114	0.6391	FALSE	8	21	22	77	0.266667	0.275862
31	D107	0.508272	FALSE	8	21	23	76	0.258065	0.275862
32	D089	0.474764	FALSE	8	21	24	75	0.25	0.275862

Table 5.11 Experiment 1 results (Precision & Recall) WGET[33..64]

WGET	Documents	Weight	Relevant	R-R	R-N	I-R	I-N	Precision	Recall
33	D078	0.477959	FALSE	8	21	25	74	0.242424	0.275862
34	D075	0.537479	FALSE	8	21	26	73	0.235294	0.275862
35	D105	0.637283	FALSE	8	21	27	72	0.228571	0.275862
36	D032	0.578318	FALSE	8	21	28	71	0.222222	0.275862
37	D057	0.68649	TRUE	9	20	28	71	0.243243	0.310345
38	D036	0.681048	TRUE	10	19	28	71	0.263158	0.344828
39	D127	0.877714	TRUE	11	18	28	71	0.282051	0.37931
40	D073	0.616113	FALSE	11	18	29	70	0.275	0.37931
41	D085	0.353067	FALSE	11	18	30	69	0.268293	0.37931
42	D097	0.859593	TRUE	12	17	30	69	0.285714	0.413793
43	D081	0.779926	TRUE	13	16	30	69	0.302326	0.448276
44	D037	0.605731	FALSE	13	16	31	68	0.295455	0.448276
45	D104	0.670373	FALSE	13	16	32	67	0.288889	0.448276
46	D033	0.517563	FALSE	13	16	33	66	0.282609	0.448276
47	D049	0.554995	FALSE	13	16	34	65	0.276596	0.448276
48	D111	0.633138	FALSE	13	16	35	64	0.270833	0.448276
49	D044	0.6033	FALSE	13	16	36	63	0.265306	0.448276
50	D077	0.658674	FALSE	13	16	37	62	0.26	0.448276
51	D023	0.592686	FALSE	13	16	38	61	0.254902	0.448276
52	D050	0.564782	FALSE	13	16	39	60	0.25	0.448276
53	D026	0.612502	FALSE	13	16	40	59	0.245283	0.448276
54	D027	0.471359	FALSE	13	16	41	58	0.240741	0.448276
55	D063	0.575586	FALSE	13	16	42	57	0.236364	0.448276
56	D029	0.570268	FALSE	13	16	43	56	0.232143	0.448276
57	D128	0.693741	TRUE	14	15	43	56	0.245614	0.482759
58	D121	0.450292	FALSE	14	15	44	55	0.241379	0.482759
59	D003	0.625641	FALSE	14	15	45	54	0.237288	0.482759
60	D091	0.473705	FALSE	14	15	46	53	0.233333	0.482759
61	D030	0.593542	FALSE	14	15	47	52	0.229508	0.482759
62	D098	0.82863	TRUE	15	14	47	52	0.241935	0.517241
63	D004	0.674718	FALSE	15	14	48	51	0.238095	0.517241
64	D103	0.771209	TRUE	16	13	48	51	0.25	0.551724

Table 5.12 Experiment 1 results (Precision & Recall) WGET[65..96]

WGET	Documents	Weight	Relevant	R-R	R-N	I-R	I-N	Precision	Recall
65	D109	0.739249	TRUE	17	12	48	51	0.261538	0.586207
66	D125	0.837158	TRUE	18	11	48	51	0.272727	0.62069
67	D039	0.638393	FALSE	18	11	49	50	0.268657	0.62069
68	D058	0.558378	FALSE	18	11	50	49	0.264706	0.62069
69	D001	0.67094	FALSE	18	11	51	48	0.26087	0.62069
70	D113	0.834881	TRUE	19	10	51	48	0.271429	0.655172
71	D093	0.727403	TRUE	20	9	51	48	0.28169	0.689655
72	D017	0.643965	FALSE	20	9	52	47	0.277778	0.689655
73	D051	0.602936	FALSE	20	9	53	46	0.273973	0.689655
74	D116	0.637703	FALSE	20	9	54	45	0.27027	0.689655
75	D067	0.620184	FALSE	20	9	55	44	0.266667	0.689655
76	D115	0.605136	FALSE	20	9	56	43	0.263158	0.689655
77	D061	0.658942	FALSE	20	9	57	42	0.25974	0.689655
78	D002	0.609397	FALSE	20	9	58	41	0.25641	0.689655
79	D092	0.413948	FALSE	20	9	59	40	0.253165	0.689655
80	D007	0.632233	FALSE	20	9	60	39	0.25	0.689655
81	D074	0.665074	FALSE	20	9	61	38	0.246914	0.689655
82	D038	0.620836	FALSE	20	9	62	37	0.243902	0.689655
83	D083	0.523528	FALSE	20	9	63	36	0.240964	0.689655
84	D122	0.890193	TRUE	21	8	63	36	0.25	0.724138
85	D046	0.635419	FALSE	21	8	64	35	0.247059	0.724138
86	D124	0.745513	TRUE	22	7	64	35	0.255814	0.758621
87	D055	0.446511	FALSE	22	7	65	34	0.252874	0.758621
88	D119	0.33586	FALSE	22	7	66	33	0.25	0.758621
89	D101	0.592929	FALSE	22	7	67	32	0.247191	0.758621
90	D120	0.629868	FALSE	22	7	68	31	0.244444	0.758621
91	D102	0.729473	TRUE	23	6	68	31	0.252747	0.793103
92	D088	0.649748	FALSE	23	6	69	30	0.25	0.793103
93	D025	0.555248	FALSE	23	6	70	29	0.247312	0.793103
94	D021	0.639761	FALSE	23	6	71	28	0.244681	0.793103
95	D010	0.504824	FALSE	23	6	72	27	0.242105	0.793103
96	D096	0.739996	TRUE	24	5	72	27	0.25	0.827586

Table 5.13 Experiment 1 results (Precision & Recall) WGET[97..128]

WGET	Documents	Weight	Relevant	R-R	R-N	I-R	I-N	Precision	Recall
97	D126	0.647634	FALSE	24	5	73	26	0.247423	0.827586
98	D099	0.629595	FALSE	24	5	74	25	0.244898	0.827586
99	D053	0.608683	FALSE	24	5	75	24	0.242424	0.827586
100	D110	0.547243	FALSE	24	5	76	23	0.24	0.827586
101	D034	0.681417	TRUE	25	4	76	23	0.247525	0.862069
102	D059	0.616141	FALSE	25	4	77	22	0.245098	0.862069
103	D080	0.632831	FALSE	25	4	78	21	0.242718	0.862069
104	D022	0.590105	FALSE	25	4	79	20	0.240385	0.862069
105	D112	0.529016	FALSE	25	4	80	19	0.238095	0.862069
106	D123	0.317798	FALSE	25	4	81	18	0.235849	0.862069
107	D060	0.674157	FALSE	25	4	82	17	0.233645	0.862069
108	D011	0.604205	FALSE	25	4	83	16	0.231481	0.862069
109	D086	0.652306	FALSE	25	4	84	15	0.229358	0.862069
110	D087	0.657575	FALSE	25	4	85	14	0.227273	0.862069
111	D009	0.557666	FALSE	25	4	86	13	0.225225	0.862069
112	D020	0.685407	TRUE	26	3	86	13	0.232143	0.896552
113	D066	0.787155	TRUE	27	2	86	13	0.238938	0.931034
114	D016	0.506173	FALSE	27	2	87	12	0.236842	0.931034
115	D013	0.639832	FALSE	27	2	88	11	0.234783	0.931034
116	D100	0.521291	FALSE	27	2	89	10	0.232759	0.931034
117	D084	0.30538	FALSE	27	2	90	9	0.230769	0.931034
118	D094	0.61339	FALSE	27	2	91	8	0.228814	0.931034
119	D065	0.47012	FALSE	27	2	92	7	0.226891	0.931034
120	D040	0.595716	FALSE	27	2	93	6	0.225	0.931034
121	D005	0.728029	TRUE	28	1	93	6	0.231405	0.965517
122	D064	0.532658	FALSE	28	1	94	5	0.229508	0.965517
123	D118	0.584407	FALSE	28	1	95	4	0.227642	0.965517
124	D042	0.596187	FALSE	28	1	96	3	0.225806	0.965517
125	D019	0.698225	TRUE	29	0	96	3	0.232	1
126	D117	0.393035	FALSE	29	0	97	2	0.230159	1
127	D095	0.590978	FALSE	29	0	98	1	0.228346	1
128	D048	0.548474	FALSE	29	0	99	0	0.226563	1

5.3.2 Experiment 2

The second experiment we started by applying our topic-search agent system that uses PLSI method on this collection with our filters and thresholds inside it. Then we calculated precision and recall too. The details will be presented later in the next sections.

We have selected our threshold on document weight to be 0.6 to be able to pass through role C. This means that some of the documents will be ignored and not be send to the store volume of relevant documents.

Then we calculated precision and recall. The details are as shown in Tables 5.14 and 5.15. Note that the sequence of documents to be fetched was according to the links queue and those are inside the documents starting from Document 82.

Table 5.14 Experiment 2 results (Precision & Recall) WGET[1..50]

WGET	Documents	Weight	Relevant	R-R	R-N	I-R	I-N	Precision	Recall
1	D082	0.504781	FALSE	0	29	1	98	0	0
2	D045	0.682782	TRUE	1	28	1	98	0.5	0.034483
3	D035	0.692718	TRUE	2	27	1	98	0.666667	0.068966
4	D062	0.616768	FALSE	2	27	2	97	0.5	0.068966
5	D106	0.744729	TRUE	3	26	2	97	0.6	0.103448
6	D072	0.713054	TRUE	4	25	2	97	0.666667	0.137931
7	D043	0.60354	FALSE	4	25	3	96	0.571429	0.137931
8	D108	0.655617	FALSE	4	25	4	95	0.5	0.137931
9	D014	0.660878	FALSE	4	25	5	94	0.444444	0.137931
10	D070	0.879918	TRUE	5	24	5	94	0.5	0.172414
11	D090	0.763442	TRUE	6	23	5	94	0.545455	0.206897
12	D052	0.604035	FALSE	6	23	6	93	0.5	0.206897
13	D028	0.737799	TRUE	7	22	6	93	0.538462	0.241379
14	D018	0.707376	TRUE	8	21	6	93	0.571429	0.275862
15	D068	0.820505	FALSE	8	21	7	92	0.533333	0.275862
16	D012	0.60342	FALSE	8	21	8	91	0.5	0.275862
17	D041	0.624845	FALSE	8	21	9	90	0.470588	0.275862
18	D114	0.6391	FALSE	8	21	10	89	0.444444	0.275862
19	D105	0.637283	FALSE	8	21	11	88	0.421053	0.275862
20	D057	0.68649	TRUE	9	20	11	88	0.45	0.310345
21	D036	0.681048	TRUE	10	19	11	88	0.47619	0.344828
22	D127	0.877714	TRUE	11	18	11	88	0.5	0.37931
23	D073	0.616113	FALSE	11	18	12	87	0.478261	0.37931
24	D097	0.859593	TRUE	12	17	12	87	0.5	0.413793
25	D081	0.779926	TRUE	13	16	12	87	0.52	0.448276
26	D037	0.605731	FALSE	13	16	13	86	0.5	0.448276
27	D104	0.670373	FALSE	13	16	14	85	0.481481	0.448276
28	D111	0.633138	FALSE	13	16	15	84	0.464286	0.448276
29	D044	0.6033	FALSE	13	16	16	83	0.448276	0.448276
30	D077	0.658674	FALSE	13	16	17	82	0.433333	0.448276
31	D026	0.612502	FALSE	13	16	18	81	0.419355	0.448276
32	D128	0.693741	TRUE	14	15	18	81	0.4375	0.482759
33	D003	0.625641	FALSE	14	15	19	80	0.424242	0.482759
34	D098	0.82863	TRUE	15	14	19	80	0.441176	0.517241
35	D004	0.674718	FALSE	15	14	20	79	0.428571	0.517241
36	D103	0.771209	TRUE	16	13	20	79	0.444444	0.551724
37	D109	0.739249	TRUE	17	12	20	79	0.459459	0.586207
38	D125	0.837158	TRUE	18	11	20	79	0.473684	0.62069
39	D039	0.638393	FALSE	18	11	21	78	0.461538	0.62069
40	D001	0.67094	FALSE	18	11	22	77	0.45	0.62069
41	D113	0.834881	TRUE	19	10	22	77	0.463415	0.655172
42	D093	0.727403	TRUE	20	9	22	77	0.47619	0.689655
43	D017	0.643965	FALSE	20	9	23	76	0.465116	0.689655
44	D051	0.602936	FALSE	20	9	24	75	0.454545	0.689655
45	D116	0.637703	FALSE	20	9	25	74	0.444444	0.689655
46	D067	0.620184	FALSE	20	9	26	73	0.434783	0.689655
47	D115	0.605136	FALSE	20	9	27	72	0.425532	0.689655
48	D061	0.658942	FALSE	20	9	28	71	0.416667	0.689655
49	D002	0.609397	FALSE	20	9	29	70	0.408163	0.689655
50	D007	0.632233	FALSE	20	9	30	69	0.4	0.689655

Table 5.15 Experiment 2 results (Precision & Recall) WGET[51..76]

WGET	Documents	Weight	Relevant	R-R	R-N	I-R	I-N	Precision	Recall
51	D074	0.665074	FALSE	20	9	31	68	0.392157	0.689655
52	D038	0.620836	FALSE	20	9	32	67	0.384615	0.689655
53	D122	0.890193	TRUE	21	8	32	67	0.396226	0.724138
54	D046	0.635419	FALSE	21	8	33	66	0.388889	0.724138
55	D124	0.745513	TRUE	22	7	33	66	0.4	0.758621
56	D120	0.629868	FALSE	22	7	34	65	0.392857	0.758621
57	D102	0.729473	TRUE	23	6	34	65	0.403509	0.793103
58	D088	0.649748	FALSE	23	6	35	64	0.396552	0.793103
59	D021	0.639761	FALSE	23	6	36	63	0.389831	0.793103
60	D096	0.739996	TRUE	24	5	36	63	0.4	0.827586
61	D126	0.647634	FALSE	24	5	37	62	0.393443	0.827586
62	D099	0.629595	FALSE	24	5	38	61	0.387097	0.827586
63	D053	0.608683	FALSE	24	5	39	60	0.380952	0.827586
64	D034	0.681417	TRUE	25	4	39	60	0.390625	0.862069
65	D059	0.616141	FALSE	25	4	40	59	0.384615	0.862069
66	D080	0.632831	FALSE	25	4	41	58	0.378788	0.862069
67	D060	0.674157	FALSE	25	4	42	57	0.373134	0.862069
68	D011	0.604205	FALSE	25	4	43	56	0.367647	0.862069
69	D086	0.652306	FALSE	25	4	44	55	0.362319	0.862069
70	D087	0.657575	FALSE	25	4	45	54	0.357143	0.862069
71	D020	0.685407	TRUE	26	3	45	54	0.366197	0.896552
72	D066	0.787155	TRUE	27	2	45	54	0.375	0.931034
73	D013	0.639832	FALSE	27	2	46	53	0.369863	0.931034
74	D094	0.61339	FALSE	27	2	47	52	0.364865	0.931034
75	D005	0.728029	TRUE	28	1	47	52	0.373333	0.965517
76	D019	0.698225	TRUE	29	0	47	52	0.381579	1

Note that only seventy-six document succeeded to pass the threshold and filters, and gave better results in precision, recall and speed.

But, when we tried to increase our threshold to 0.7, the precision increases but did not recall about 72% of the relevant documents. As shown in table 5.16

Table 5.16 Experiment 3 results (Precision & Recall) WGET[1..23]

WGET	Documents	Weight	Relevant	R-R	R-N	I-R	I-N	Precision	Recall
1	D082	0.504781	FALSE	0	29	1	98	0	0
2	D106	0.744729	TRUE	1	28	1	98	0.5	0.034483
3	D072	0.713054	TRUE	2	27	1	98	0.666667	0.068966
4	D070	0.879918	TRUE	3	26	1	98	0.75	0.103448
5	D090	0.763442	TRUE	4	25	1	98	0.8	0.137931
6	D028	0.737799	TRUE	5	24	1	98	0.833333	0.172414
7	D018	0.707376	TRUE	6	23	1	98	0.857143	0.206897
8	D068	0.820505	FALSE	6	23	2	97	0.75	0.206897
9	D127	0.877714	TRUE	7	22	2	97	0.777778	0.241379
10	D097	0.859593	TRUE	8	21	2	97	0.8	0.275862
11	D081	0.779926	TRUE	9	20	2	97	0.818182	0.310345
12	D098	0.82863	TRUE	10	19	2	97	0.833333	0.344828
13	D103	0.771209	TRUE	11	18	2	97	0.846154	0.37931
14	D109	0.739249	TRUE	12	17	2	97	0.857143	0.413793
15	D125	0.837158	TRUE	13	16	2	97	0.866667	0.448276
16	D113	0.834881	TRUE	14	15	2	97	0.875	0.482759
17	D093	0.727403	TRUE	15	14	2	97	0.882353	0.517241
18	D122	0.890193	TRUE	16	13	2	97	0.888889	0.551724
19	D124	0.745513	TRUE	17	12	2	97	0.894737	0.586207
20	D102	0.729473	TRUE	18	11	2	97	0.9	0.62069
21	D096	0.739996	TRUE	19	10	2	97	0.904762	0.655172
22	D066	0.787155	TRUE	20	9	2	97	0.909091	0.689655
23	D005	0.728029	TRUE	21	8	2	97	0.913043	0.724138

5.3.3 Experiment 3

We have repeated all the previously mentioned steps to another collection of documents that part of them deals with medical-issues. Medical terms were used in the input query.

We have collected a set of medical related documents with the help of specialists in medical issues. The documents collection contained also non-relevant documents. The specialists helped us in filling the input query which should help the agent in building his core index (knowledge) about such topic.

Two tests on the collection were made:-

- 1- Testing the system with our contributed filter and threshold.
- 2- Testing the system without (filter and threshold).

In each of those tests we monitored the system output results and recoded all readings came out in precision and recall after every document fetched from the collection.

This experiment was important for me to compare precision and recall between two different topics. We 'v calculated the recall and precision of the system output with and without filter and threshold in order to note difference between topics in multi-agent system for more than one topic.

The main objective from implementing this experiment is to calculate precision and recall using our contributed system and to note its behavior. We discovered that, on one hand, the system gave better results in precision and recall form other systems we have described before.

On the other hand, the system gave better results in precision and recall from other topic we have used before which is "computer science".

The different between those two topics will be discussed in more details in the next section.

5.4 Summary & Conclusion

Testing and implementation for such system is somehow complicated. We need useful tools for implementation. Plus, designing and programming such agent with no Agent programming language is difficult. So we tried to program our own system with the most required classes and methods which we mentioned before and used in experiments.

A part of our future work we recommend on this system is to complete building the agent using an agent programming tools that have been made in the recent years like JADE (Java Agent Development) or any other similar programming tools.

Comparing the results of these experiments indicated progress in precision and recall of our architecture and design. In chapter six, we will discuss in details the advantages of our system.

From results of experiment one and two, we can see that our filters and thresholds increase the efficiency and the speed of our system. If we draw a chart figures for both experiments as shown for experiment one in Figure 5.8 and experiment two in Figure 5.9, we will note that precision is kept in a high, stable and reliable values, on the other hand the system succeeded to recall all relevant document without fetching a lot of non-relevant documents.

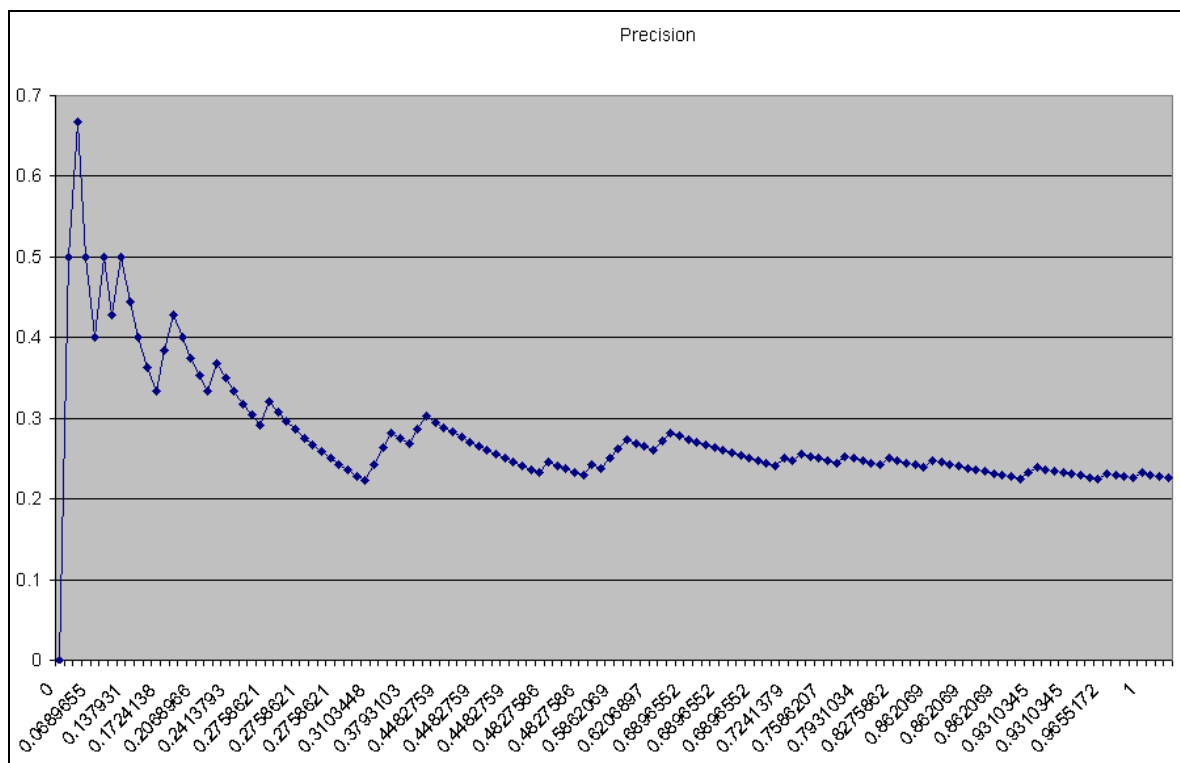


Figure 5.8 Experiment 1 results chart. (traditional system)

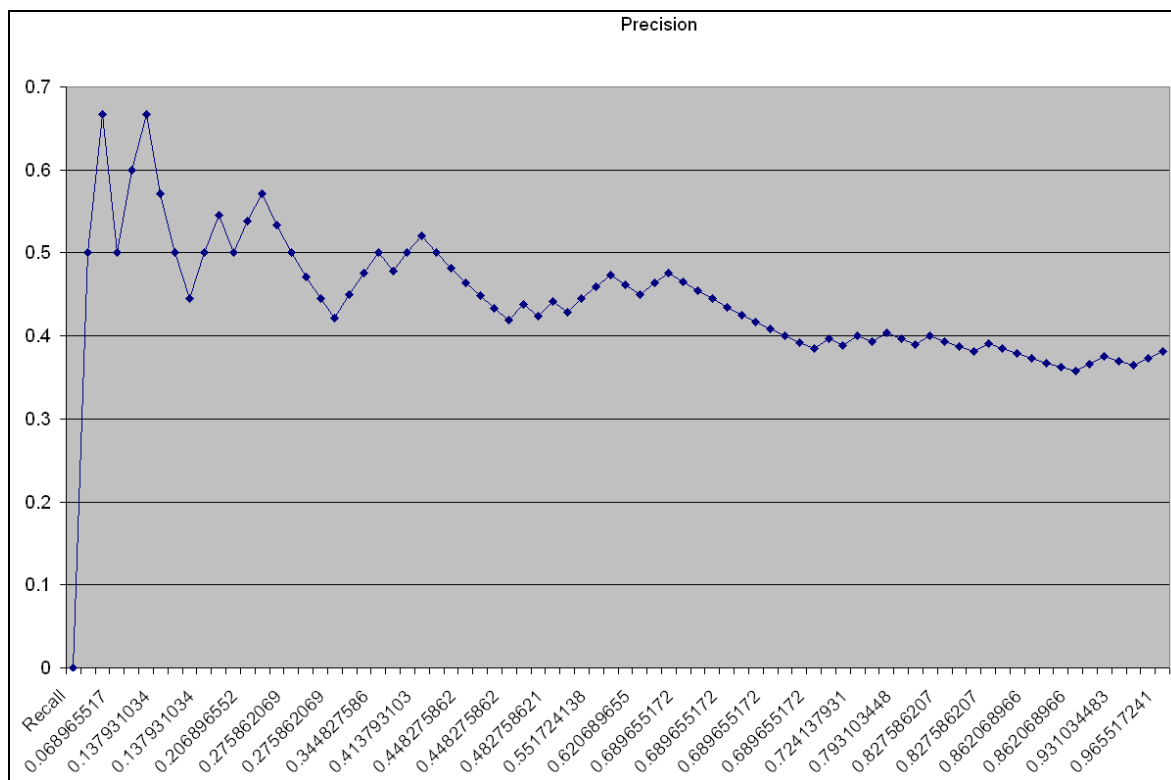


Figure 5.9 Experiment 2 results chart. (Our System)

Selecting documents for the test-collection for experiments 1 and 2 which was made on "computer science" topic was handled by us. This was our starting topic, because we consider our-selves familiar with such topic and have the ability to decide which document is relevant among all documents.

Results that have been recorded in precision and recall in both experiments 1 and 2 was very helpful in proving that our contributed filter and threshold were successful in giving better results in precision and recall on one hand. But on the other hand, it did affect the speed of the system, which is the factor that because it is not one of the requirements to this system has been ignored in our agent system,

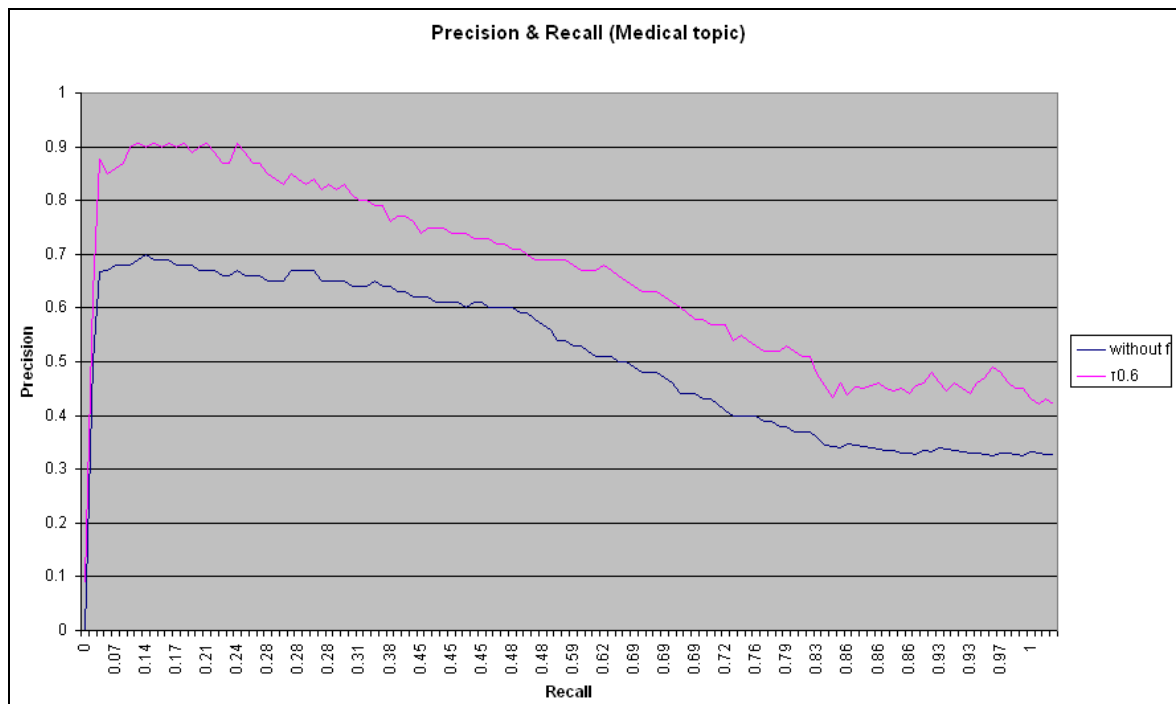


Figure 5.10 Experiment 3 Precision & Recall on Medical topic

Figure 5.10, shows the implementation of another agent applied to another topic. The new topic was (medical topic). We wanted to see the system behavior on completely different topic in order to get some notes and imagine this agent in a multi-agent system.

We got the following notes:-

- Precision in (Medical topic) is much higher than (Computer science topic).
- Medical terms in most cases are very unique and very strong key words and gives high results.
- Our agent – system (with filter and threshold = 0.6) gives higher precision. But not as much as in (Computer science).

In "Computer science" topic, a lot of terms that is considered to be related to is related to other topics at the same time but with different weight. These weights are sometimes having only small differences in between, which is not in "medical issues". Let's take for example the term "treatment" its weight in "medical topic" was recorded about (0.6321201) but in "computer science" was about (0.5214.286). Such term is not absolutely concerned to be only a medical topic term, it can be a "computer science" topic related too. As well as "virus", "infection" ... etc. but terms like "medicine", "Aspirin", "cholera" etc, have extremely high weights in medical topics.

Chapter 6

Conclusions and Future works.

6.1 Conclusions

In this chapter we are going to discuss the two experiments which were applied to our agent system, concentrating on one topic. The first experiment used traditional PLSI method with no filter, while the other experiment was with filter. Then we got results from each one and note the difference between them.

We will also analyze, summarize and present the results of the experiments. Then, our conclusions and further work will be presented too.

From results of experiment one and two, we can see that our filters and thresholds increase the efficiency and the speed of our system. If we draw a chart figures for both experiments as shown for experiment one in Figure 5.8 and experiment two in Figure 5.9, we will note that precision is kept in a high, stable and reliable values, on the other hand the system succeeded to recall all relevant document without fetching a lot of non-relevant documents.

Testing and implementation for such system was not that easy. Preparing useful tools in implementation, in addition to designing and programming such agent with no Agent programming language is difficult. So we have programmed our own tool with the most required classes and methods which we mentioned before and used in experiments.

As we see that the results of both experiments indicate a progress in the precision and recall of our architecture and design. In this chapter, we will discuss in details the advantages of our system.

As mentioned before. In testing our architecture, we've used a collection on the information retrieval topic. The collection core contains many documents which were selected by topic administrator (Agent programming topic) relevant documents.

These documents were used to generate the collection topic analyzer. We've used a set of terms as topic-terms archive. These terms were supplied by an administrator also and were used for topic-analyzer generation.

Agent starts with a set of start Links. This set contains about 5 links on relevant html documents and was presented by the administrator.

From results of experiment one and two, we can see that our filters and thresholds increase the precision and recall of our system. If we draw a chart figures for both experiments as shown for experiment one in Figure 5.8 and experiment two in Figure 5.9, we note that precision is kept higher and better than traditional system in all the test range from 0% to 100% of document collection, in addition, the system succeeded to recall all relevant document without fetching a lot of non-relevant documents.

By monitoring and calculating precision and recall after every single document fetched from the collection until the end of the collection, note that our filter blocked most of non-relevant documents and not allowing them to pass.

Assume that document d_x is related to our topic with weight $w = 0.6$. this document will manage to pass through threshold $\tau \leq 0.6$. but if the threshold increases to $\tau > 0.6$, the document d_x will not manage to pass and will be abandoned by agent role C.

For sure increasing τ will increase precision, but this will affect recall in a negative way. In other words, some of relevant documents to the system will not manage to pass through the thresholds with weight $w > \tau$. This means that the owner should choose in particular point between precision and recall. In other words, he should decide the target of his agent by valuing the threshold volume.

Note that if we take the whole range of $\tau: [0,1]$ and tried to monitor precision and recall we will note the following:

- For $\tau = 0$, system is working as traditional search system with no filter.
- For $\tau = 1$, system will bring almost no document but those with $w = \tau$.
- For $1 > \tau > 0$, system will select document according to its relevance to topic.

The goal of the second experiment test was to find best values for thresholds for document evaluator and topic-Analyzer. Result of the test shows that in the best case the precision of the agent is 0.722. This means that our agent downloaded a set of new documents, about 72 % of which were recommended by this agent to the collection, and our topic-administrator estimate that almost all recommended documents are relevant to the collection topic. As shown in Figure 6.1 and Figure 6.2.

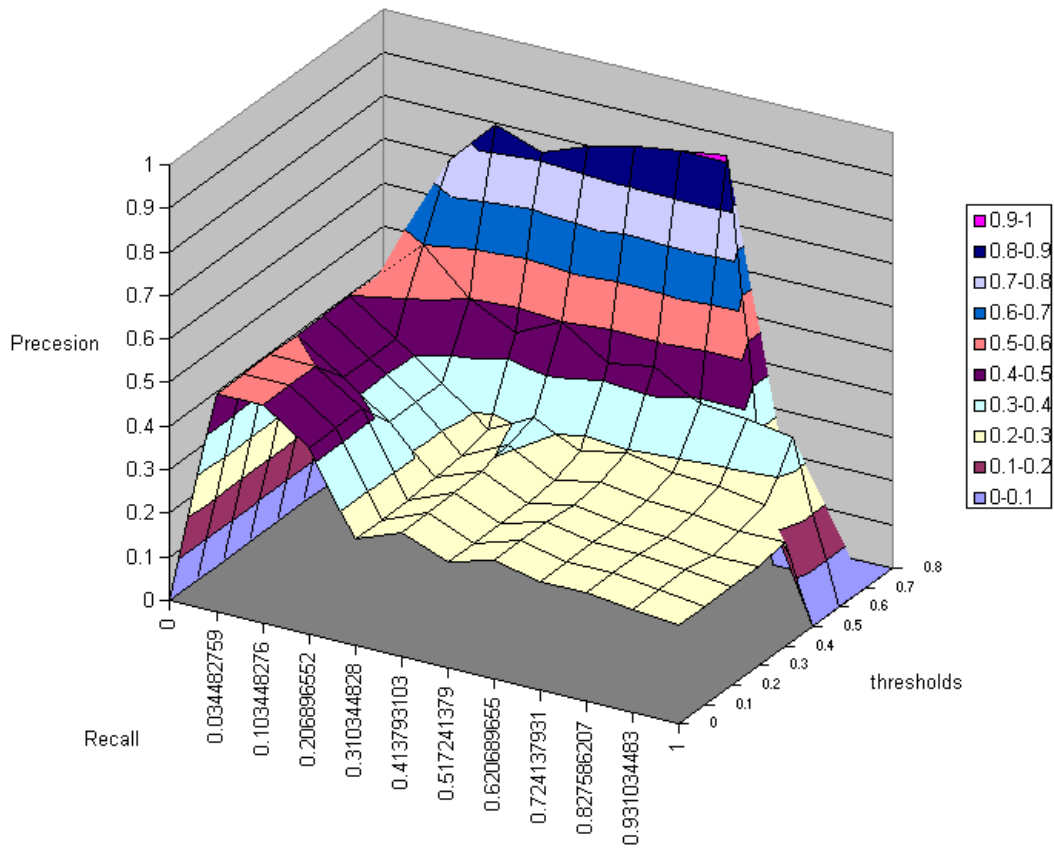


Figure 6.1 surface chart for best threshold

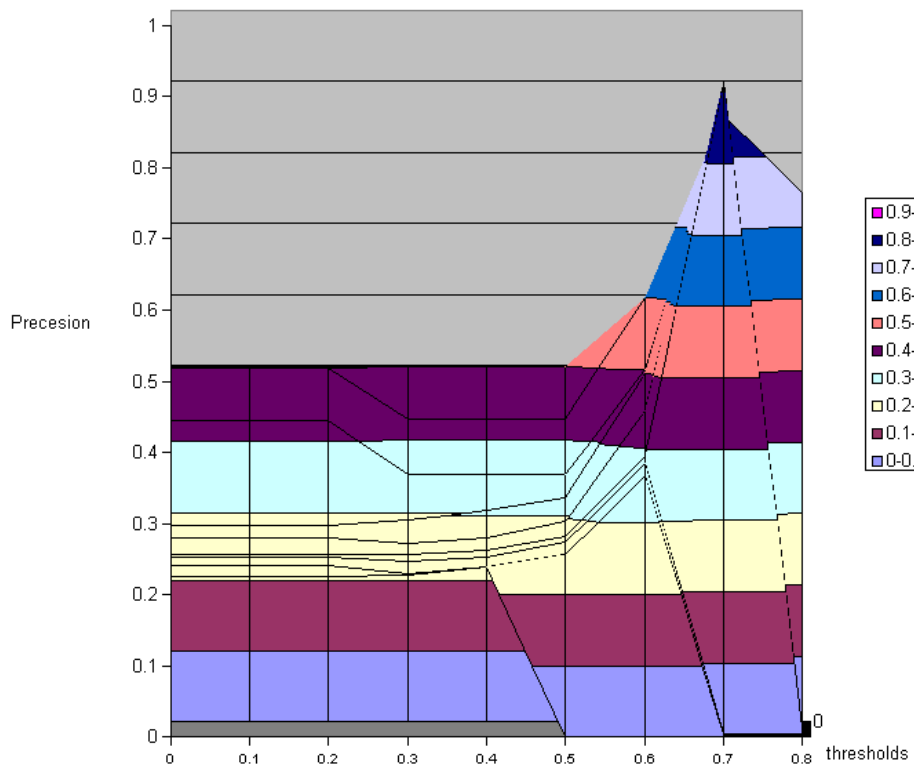


Figure 6.2 precision according to threshold τ

6.2 Future works

It was important to implement experiments on our system without filter and record results and to compare with results that have been achieved by other researchers in similar systems that use similar PLSI method. Not only similar method but also similar to our selected-topic. Since we found out that results are almost the same, then it was the suitable environment to test our filter and threshold.

As we can see that, selecting the best threshold is not an easy job. It depends on the topic we are searching for. And it depends also on the user himself, and how many documents to be retrieved to his own collection.

Finally, it was not possible to program a complete agent using Java language to satisfy the requirements. We programmed small separate tools in Java to complete all roles of the agent.

Working in this agent will not stop. We hope in the future to continue improving such agent using an agent programming language, which could help a lot to make it more powerful, fast and intelligent.

By the end of the thesis we recommend the following future work:

- This agent should be rebuilding using AOP agent oriented programming language. JADE for example might be helpful in designing such system.
- Developing Agent role A, to be capable of dealing with other languages like Arabic or any other language in addition to English. This will need more studies in stemming algorithms. The agent could be a Multilanguage topic searcher. And this will increase his influence in environment.
- Developing the mathematical method of indexing core terms of target topic. In addition to indexing terms of fetched documents too.
- Implementing more studies on agent role D and selecting or contributing more effective mathematical method to indexing links inside relevant document and selecting more relevant links to be put in the queue, which will make a continuous and dynamic re-indexing of links inside the queue.
- Developing a better mathematical formula to re-index the terms of the core index terms taking advantage from every new succeeded document that manage to pass through filter, in a dynamic way.

References

- [1] Survey On Web Information Retrieval Technologies, Lan Huang, Computer Science Department, State University of New York at Stony Brook, 2000.
- [2] Efficient crawling through url ordering, Junghoo Cho and Lawrence Page, Proceedings of the 8th International WWW conference, Canada, Tronoto, May 1999.
- [3] Multi-agent system of a distributed environment to build thematic collections Rushdi Hamamreh. // Web Instruments, № 9, 2001.
- [4] Intelligent information Agents, S. Haverkamp, JASIS, V49, N4, 1998.
- [5] Latent class models for collaborative filtering, Hofmann T.,. In Proceedings of the 16th International Joint conference on Artificial Intelligent, 1999.
- [6] An introduction to Multiagent Systems, Michael Wooldridge, John Wiley & sons Ltd, 2002.
- [7] Multi-agent Systems A Modern Approach to Distributed Modern Approach to Artificial Intelligence, Gerhard Weiss, The MIT Press, Cambridge, Massachusetts, Massachusetts Institute of Technology.Lan, London, England,©1999,
- [8] Towards Agent Societies for Information Retrieval, Holger Billhardt and Sascha Ossowski Universidad Rey Juan Carlos Dpt. of Computer Science. 2001.
- [9] World Internet Usage And Population Statistics, IWS (Internet world stats) website www.internetworldstats.com.
- [10] Agent Role Locking Theory ARL, Salaheddin J. Juneidi. MIT-Press Cambridge, USA, 1-3 Nov. 2005.
- [11] Toward Programming Paradigms For Agent Oriented Software Engineering,. Salaheddin J. Juneidi IASTED SE Austria. 18-20 feb. 2004.
- [12] Agent orientation in software engineering, Gerhard Weib, Cambridge University. 2001.
- [13] Agent-oriented software engineering, Carolre Bernon, Massimo Cossentino and Juan Pavon, Cambridge University, 2005.
- [14] Supporting Agent-Oriented Modelling With UML, Federico Bergenti and Agostino Poggi, University degli Studi di Parma. International Journal of Software Engineering and Knowledge Engineering Vol. 12, No. 6 605{618) 2002.
- [15] Evaluation of Agent Oriented Software Engineering, Main Approaches, Salaheddin J. Juneidi, George A. Vouros, University of the Aegean.
- [16] Using Java for Artificial Intelligence and Intelligent Agent Systems, October 1999, Paolo Busetta, Ralph Rönquist, Andrew Hodgson & Andrew Lucas, Agent Oriented Software Pty. Ltd., Melbourne, Australia
- [17] Autonomous Agent For Gathering Information To Build Focused Index From Distributed Environment, Rushdi A. Hamamreh, IJCSNS, Jan. 2008.
- [18] Cooperative Multi-Agent Information Gathering, Keith Decker, Victor Lesser, M. V. NagendraPrasad, and Thomas Wagner.
- [19] Designing a Multi-Agent Portfolio Management System, Keith Decker, Katia Sycara, and Dajun Zeng, The Robotics Institute, Carnegie Mellon University.
- [20] Extending a Multi-Agent System for Genomic Annotation, Keith Decker, Salim Khan, Carl Schmidt, and Dennis Michaud, Computer and Information Sciences Department, University of Delaware.
- [21] MultiAgent Integration of Information Gathering and Decision Support Katia Sycara and Dajun Zeng, Published by John Wiley & Sons, Ltd. in 1996.
- [22] Multiagent Planning for Agents with Internal Execution Resource Constraints, Haksun Li, Edmund H. Durfee, Kang G. Shin, The University of Michigan, Ann Arbor, Copyright 2003.

- [23] Building A Modern Standard Arabic Corpus, Ahmed Abdelali, Jim Cowie, Hamdy S. Soliman, New Mexico State University.
- [24] Distributed Search-Based Advertising On The Web, Nikita Schmidt and Ahmed Patel, Saint Petersburg State University. 1999.
- [25] Modern information retrieval, Ricardo Christopher D. Manning Prabhakar Raghavan Hinrich Schütze 2008.
- [26] Information Retrieval Algorithms and Heuristics, David A. Grossman, Opher Frieder, Second edition. 2004.
- [27] Probabilistic Latent Semantic Indexing, Thomas Hofmann, International Computer Science Institute, Berkeley, CA & EECS Department, CS Division, UC Berkeley.
- [28] Towards an Architecture for A-life Agents, Darryl N. Davis, T. Chalabi and B. Berbank-Green, Stanford University

```
import javax.swing.UIManager;
import java.awt.*;

public class AgentA
{
    boolean packFrame = false;

    /**Construct the application*/
    public AgentA()
    {
        InterFacel frame = new InterFacel();
        //Validate frames that have preset sizes
        //Pack frames that have useful preferred size info, e.g. from their layout
        if (packFrame)
        {
            frame.pack();
        }
        else
        {
            frame.validate();
        }
        //Center the window
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        Dimension frameSize = frame.getSize();
        if (frameSize.height > screenSize.height)
        {
            frameSize.height = screenSize.height;
        }
        if (frameSize.width > screenSize.width)
        {
            frameSize.width = screenSize.width;
        }
        frame.setLocation((screenSize.width - frameSize.width) / 2, (screenSize.height - frameSize.height) / 2);
        frame.setVisible(true);
    }
}
```

Figure 7.1 AgentA Class (user interface)

```
/**Main method*/
public static void main(String[] args)
{
    try
    {
        UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    }
    catch(Exception e)
    {
        e.printStackTrace();
    }
    new AgentA();
} //End Main Method

} // End Class AgentA
```

Figure 7.2 Agent's Main Method

```

//*****
public class InterFacel extends JFrame
{
    JPanel contentPane;
    BorderLayout BorderLayout1 = new BorderLayout();
    JTextField WebURL = new JTextField(40); // Text Field to enter web address to get
    JTextField inputField = new JTextField(40); // Text Field to input query terms(topic)
    JTextArea display = new JTextArea( 15, 20 ); // To Display website terms and frequencies
    JTextArea qdisplay = new JTextArea( 15, 20 ); // To Display topic terms and values
    JTextArea cdisplay = new JTextArea( 15, 20 ); // To Display topic terms and frequencies
    JScrollPane displayScrollPane = new JScrollPane( display );
    JScrollPane qdisplayScrollPane = new JScrollPane( qdisplay );
    JScrollPane cdisplayScrollPane = new JScrollPane( cdisplay );
    String term, query;
    Hashtable table = new Hashtable(); // file terms
    Hashtable qtable = new Hashtable(); // topic terms
    Hashtable ctable = new Hashtable(); // topic-file terms
//*****
    /**Construct the frame*/
    public InterFacel()
    {
        enableEvents(AWTEvent.WINDOW_EVENT_MASK);
        try
        {
            jbInit();
        } // End try
        catch(Exception e)
        {
            e.printStackTrace();
        } // End catch(Exception e)
    } // End InterFacel()
//*****

```

Figure 7.3 Interfacel class (Frame constructor)

```

//*****
// create table from user input
private void createTable()
{
    String input = term;
    StringTokenizer words = new StringTokenizer( input, " \n\t\r" );

    while ( words.hasMoreTokens() )
    {
        String word = words.nextToken().toLowerCase(); // get word
        if ( table.containsKey( word ) ) // if the table contains the word
        {
            Integer count = (Integer) table.get( word ); // get value (integer number)
            table.put( word, new Integer( count.intValue() + 1 ) ); // and increment it
        } //End If
        else // otherwise add the word with a value of 1
            table.put( word, new Integer( 1 ) );
        if ( ctable.containsKey( word ) ) // if the query table contains the word
        {
            Integer count = (Integer) ctable.get( word ); // get value (integer number)
            ctable.put( word, new Integer( count.intValue() + 1 ) ); // and increment it
        } //End If
    } // end while
} // end method createTable
//*****

```

Figure 7.4 Dealing with user input.

```

//*****
// create qtable from user input (InputFeild)
private void qcreateTable()
{
    String input = query;
    StringTokenizer words = new StringTokenizer( input, " \n\t\r" );

    while ( words.hasMoreTokens() )
    {
        String word = words.nextToken().toLowerCase(); // get word
        if ( qtable.containsKey( word ) ) // if the table contains the word
        {
            Integer count = (Integer) qtable.get( word ); // get value (integer number)
            qtable.put( word, new Integer( count.intValue() + 1 ) ); // and increment it
        } //End If
        else // otherwise add the word with a value of 1
        {
            qtable.put( word, new Integer( 1 ) );
            ctable.put( word, new Integer( 0 ) );
        } // End else
    } // end while
} // end method createTable
//*****

```

Figure 7.5 Creating query table from user's input field

```

//*****
// create string containing table values
private String createOutput()
{
    String output = "";
    Enumeration keys = table.keys();
    // iterate through the keys
    while ( keys.hasMoreElements() )
    {
        Object currentKey = keys.nextElement();
        // output the key-value pairs
        output += currentKey + "\t" + table.get( currentKey ) + "\n";
    } // End While
    output += "size: " + table.size() + "\n";
    return output;
} // end method createOutput
//*****

```

Figure 7.6 creatOutput() method

```

//*****
// create string containing qtable values
private String qcreateOutput()
{
    String output = "";
    Enumeration keys = qtable.keys();
    // iterate through the keys
    while ( keys.hasMoreElements() )
    {
        Object currentKey = keys.nextElement();
        // output the key-value pairs
        output += currentKey + "\t" + qtable.get( currentKey ) + "\n";
    } // End While
    output += "size: " + qtable.size() + "\n";
    return output;
} // end method createOutput
//*****

```

Figure 7.7 qcreateOutput() method

```

//*****
// create string containing ctable values
private String ccreateOutput()
{
    String output = "";
    Enumeration keys = ctable.keys();
    // iterate through the keys
    while ( keys.hasMoreElements() )
    {
        Object currentKey = keys.nextElement();
        // output the key-value pairs
        output += currentKey + "\t" + ctable.get( currentKey ) + "\n";
    } // End While
    output += "size: " + ctable.size() + "\n";
    return output;
} // end method createOutput
//*****

```

Figure 7.8 ccreatOutput() method

```

//*****
/**Component initialization*/
private void jbInit() throws Exception
{
    contentPane = (JPanel) this.getContentPane();
    WebURL.setText("Enter URL");
    inputField.setText("Input Query(topic terms)");
    contentPane.setLayout(/* BorderLayout */ new FlowLayout());
    this.setSize(new Dimension(600, 450));
    this.setTitle("Agent InterFace");

    contentPane.add(WebURL/*, BorderLayout.NORTH*/);
    contentPane.add(inputField);
    contentPane.add(displayScrollPane);
    contentPane.add(qdisplayScrollPane);
    contentPane.add(cdisplayScrollPane);

    // create an instance of inner class ActionEventHandler
    ActionEventHandler handler = new ActionEventHandler();
    WebURL.addActionListener( handler );
    inputField.addActionListener( handler );

    display.setEditable( false );
    qdisplay.setEditable( false );
    cdisplay.setEditable( false );
} // End jbInit()
//*****

```

Figure 7.9 Component initialization.

```

//*****
// inner class declaration for handling JTextField and JButton events
private class ActionEventHandler implements ActionListener
{
    // method to handle action events
    public void actionPerformed( ActionEvent event )
    {
        // user pressed Enter key in WebURL
        if ( event.getSource() == WebURL )
        {
            String Web = WebURL.getText();
            String File = "Website.html";//Local File where the URL contents to be saved
            System.out.println("Agent will get site >> " + Web + " To " + File);
            try //WGET to bring the URL Contents from the internet
            {
                wget httpGetter = new wget();
                httpGetter.get( Web , File );
            } //End try
            catch (Exception ex)//Error handling while getting the URL
            {
                ex.printStackTrace();
            } //End Catch
        }
    }
}
//*****

```

Figure 7.10 bringing a document from internet


```
s.stem();
{
  String u;
  // and now, to test toString() :
  u = s.toString();
  // to test getResultBuffer(), getResultLength() :
  // u = new String(s.getResultBuffer(), 0, s.getResultLength());
  if (s.getResultLength() > 2)
    if (!u.equalsIgnoreCase("html") & !u.equalsIgnoreCase("href") &
        !u.equalsIgnoreCase("the") & !u.equalsIgnoreCase("was") &
        !u.equalsIgnoreCase("has") & !u.equalsIgnoreCase("have") &
        !u.equalsIgnoreCase("they") & !u.equalsIgnoreCase("that") &
        !u.equalsIgnoreCase("will") & !u.equalsIgnoreCase("shall") &
        !u.equalsIgnoreCase("should") & !u.equalsIgnoreCase("would") &
        !u.equalsIgnoreCase("you") & !u.equalsIgnoreCase("our") &
        !u.equalsIgnoreCase("what") & !u.equalsIgnoreCase("where") &
        !u.equalsIgnoreCase("why") & !u.equalsIgnoreCase("how") &
        !u.equalsIgnoreCase("however") & !u.equalsIgnoreCase("can") &
        !u.equalsIgnoreCase("when") & !u.equalsIgnoreCase("might") &
        !u.equalsIgnoreCase("than") & !u.equalsIgnoreCase("but"))
    {
      System.out.println(u);
      term = u;
      createTable();
      display.setText( createOutput() );
      cdisplay.setText( ccreateOutput() );
    }
  }
}
} // End s.stem
```

Figure 7.11 string's stemming before creating the table.

```
public class wget
{
    static final String FS = File.separator;

    /** This method does the actual GET
     *
     * @param theUrl The URL to retrieve
     * @param filename the local file to save to
     * @exception IOException
     */
    public void get(String theUrl, String filename) throws IOException
    {
        try
        {
            URL gotoUrl = new URL(theUrl);
            InputStreamReader isr = new InputStreamReader(gotoUrl.openStream());
            BufferedReader in = new BufferedReader(isr);

            StringBuffer sb = new StringBuffer();
            String inputLine;
            boolean isFirst = true;

            //grab the contents at the URL
            while ((inputLine = in.readLine()) != null)
            {
                sb.append(inputLine+"\r\n");
                //System.out.println(inputLine);
            }

            //write it locally
            createAFile(filename, sb.toString());
        }
        catch (MalformedURLException mue)
        {
            mue.printStackTrace();
        }
        catch (IOException ioe)
        {
            throw ioe;
        }
    }
}
```

Figure 7.12 WGET Class (Java code)

```

//creates a local file
/** Writes a String to a local file
 *
 * @param outfile the file to write to
 * @param content the contents of the file
 * @exception IOException
 */
public static void createAFile(String outfile, String content) throws IOException
{
    FileOutputStream fileoutputstream = new FileOutputStream(outfile);
    DataOutputStream dataoutputstream = new DataOutputStream(fileoutputstream);
    dataoutputstream.writeBytes(content);
    dataoutputstream.flush();
    dataoutputstream.close();
}
}
} //End Class Wget

```

Figure 7.13 createAFile method.

```

//*****
/** Stemmer, implementing the Porter Stemming Algorithm
 * The Stemmer class transforms a word into its root form. The input
 * word can be provided a character at a time (by calling add()), or at once
 * by calling one of the various stem(something) methods. */
class Stemmer
{
    private char[] b;
    private int i,           /* offset into b */
               i_end,       /* offset to end of stemmed word */
               j, k;
    private static final int INC = 50; /* unit of size whereby b is increased */
//*****
    public Stemmer()
    {
        b = new char[INC];
        i = 0;
        i_end = 0;
    } // End Stemmer()
//*****

```

Figure 7.14 Stemmer class's declaration.

```

//*****
/* Add a character to the word being stemmed. When you are finished
 * adding characters, you can call stem(void) to stem the word. */
public void add(char ch)
{
    if (i == b.length)
    {
        char[] new_b = new char[i+INC];
        for (int c = 0; c < i; c++) new_b[c] = b[c];
        b = new_b;
    } // End if
    b[i++] = ch;
} // End add(char ch)
//*****

```

Figure 7.15 add() method1 in stemmer class.

```

/*****
  /** Adds wLen characters to the word being stemmed contained in a portion
  * of a char[] array. This is like repeated calls of add(char ch), but
  * faster.    */
  public void add(char[] w, int wLen)
  {
    if (i+wLen >= b.length)
    {
      char[] new_b = new char[i+wLen+INC];
      for (int c = 0; c < i; c++) new_b[c] = b[c];
      b = new_b;
    } // End if
    for (int c = 0; c < wLen; c++) b[i++] = w[c];
  } // End add(char[] w, int wLen)
*****/

```

Figure 7.16 add() method2 in stemmer class.

```

/*****
  /** After a word has been stemmed, it can be retrieved by toString(),
  * or a reference to the internal buffer can be retrieved by getResultBuffer
  * and getResultLength (which is generally more efficient.)    */
  public String toString()
  {
    return new String(b,0,i_end);
  } // End toString()
*****/

```

Figure 7.17 toString() method in stemmer class

```

/*****
  /** Returns the length of the word resulting from the stemming process.    */
  public int getResultLength()
  {
    return i_end;
  } // End getResultLength()
*****/

```

Figure 7.18 getResultLength() method in stemmer class

```

/*****
  /** Returns a reference to a character buffer containing the results of
  * the stemming process. You also need to consult getResultLength()
  * to determine the length of the result.    */
  public char[] getResultBuffer()
  {
    return b;
  } // End getResultBuffer()
*****/

```

Figure 7.19 getResultBuffer() method in stemmer class

```

//*****
/* cons(i) is true <=> b[i] is a consonant. */
private final boolean cons(int i)
{
    switch (b[i])
    {
        case 'a': case 'e': case 'i': case 'o': case 'u': return false;
        case 'y': return (i==0) ? true : !cons(i-1);
        default: return true;
    } // End switch case
} // End cons(int i)
//*****

```

Figure 7.20 consonant cases method in stemmer class

```

//*****
/* m() measures the number of consonant sequences between 0 and j. if c is
a consonant sequence and v a vowel sequence, and <..> indicates arbitrary
presence,
    <c><v>      gives 0
    <c>vc<v>    gives 1
    <c>vcvc<v>  gives 2
    <c>vcvcvc<v> gives 3
    ....
*/
private final int m()
{
    int n = 0;
    int i = 0;
    while(true)
    {
        if (i > j) return n;
        if (! cons(i)) break; i++;
    } // End While
    i++;
    while(true)
    {
        while(true)
        {
            if (i > j) return n;
            if (cons(i)) break;
            i++;
        } // End While
        i++;
        n++;
        while(true)
        {
            if (i > j) return n;
            if (! cons(i)) break;
            i++;
        } // End While
        i++;
    } // End While
} // End m()
//*****

```

Figure 7.21 consonants counter method in stemmer class

```

/*****
  /* vowelinstem() is true <=> 0,...j contains a vowel */
  private final boolean vowelinstem()
  {
    int i; for (i = 0; i <= j; i++) if (! cons(i)) return true;
    return false;
  } // End vowelinstem()
*****/

```

Figure 7.22 vowel checker method in stemmer class

```

/*****
  /* doublec(j) is true <=> j,(j-1) contain a double consonant. */
  private final boolean doublec(int j)
  {
    if (j < 1) return false;
    if (b[j] != b[j-1]) return false;
    return cons(j);
  } // End doublec(int j)
*****/

```

Figure 7.23 double consonant checker in stemmer class

```

/*****
  /* cvc(i) is true <=> i-2,i-1,i has the form consonant - vowel - consonant
  and also if the second c is not w,x or y. this is used when trying to
  restore an e at the end of a short word. e.g.
  cav(e), lov(e), hop(e), crim(e), but
  snow, box, tray. */
  private final boolean cvc(int i)
  {
    if (i < 2 || !cons(i) || cons(i-1) || !cons(i-2)) return false;
    {
      int ch = b[i];
      if (ch == 'w' || ch == 'x' || ch == 'y') return false;
    } //End if
    return true;
  } // End cvc(int i)
*****/

```

Figure 7.24 an "e" restore method in stemmer class

```

/*****
  private final boolean ends(String s)
  {
    int l = s.length();
    int o = k-1+1;
    if (o < 0) return false;
    for (int i = 0; i < l; i++) if (b[o+i] != s.charAt(i)) return false;
    j = k-1;
    return true;
  } // End ends (String s)
*****/

```

Figure 7.25 string's end checker method in stemmer class

```

//*****
/* setto(s) sets (j+1),...k to the characters in the string s, readjusting k. */
private final void setto(String s)
{
    int l = s.length();
    int o = j+1;
    for (int i = 0; i < l; i++) b[o+i] = s.charAt(i);
    k = j+1;
} // End setto(String s)
//*****

```

Figure 7.26 setto() method in stemmer class

```

//*****
/* r(s) is used further down. */
private final void r(String s)
{
    if (m() > 0) setto(s);
} // End r(String s)
//*****

```

Figure 7.27 string creator r() method in stemmer class

```

//*****
/* stepl() gets rid of plurals and -ed or -ing. e.g.
   caresses  -> caress
   ponies    -> poni
   ties      -> ti
   caress    -> caress
   cats      -> cat
   feed      -> feed
   agreed    -> agree
   disabled  -> disable
   matting   -> mat
   mating    -> mate
   meeting   -> meet
   milling   -> mill
   messing   -> mess
   meetings  -> meet*/
private final void stepl()
{
  if (b[k] == 's')
  {
    if (ends("sses")) k -= 2; else
    if (ends("ies")) setto("i"); else
    if (b[k-1] != 's') k--;
  }//End If
  if (ends("eed"))
  {
    if (m() > 0) k--;
  }//End if
  else
  if ((ends("ed") || ends("ing")) && vowelinstem())
  {
    k = j;
    if (ends("at")) setto("ate"); else
    if (ends("bl")) setto("ble"); else
    if (ends("iz")) setto("ize"); else
    if (doublec(k))
    {
      k--;
      {
        int ch = b[k];
        if (ch == 'l' || ch == 's' || ch == 'z') k++;
      }//End k--
    }//End If
    else if (m() == 1 && cvc(k)) setto("e");
  }//End If
} //End Stepl()
//*****

```

Figure 7.28 step1 in stemming process.


```

//*****
/* step2() turns terminal y to i when there is another vowel in the stem. */
private final void step2()
{
    //if (ends("y") && vowelinstem()) b[k] = 'i';
} // End Step2()
//*****

```

Figure 7.29 step2 in stemming process

<pre> //***** /* step3() maps double suffixes to single ones. so -ization (= -ize plus -ation) maps to -ize etc. note that the string before the suffix must give m() > 0. */ private final void step3() { if (k == 0) return; /* For Bug 1 */ switch (b[k-1]) { case 'a': if (ends("ational")) { r("ate"); break; } //End If if (ends("tional")) { r("tion"); break; } //End If break; case 'c': if (ends("enci")) { r("ence"); break; } //End If if (ends("anci")) { r("ance"); break; } //End If break; case 'e': if (ends("izer")) { r("ize"); break; } //End If break; case 'l': if (ends("bli")) { r("ble"); break; } //End If if (ends("alli")) { r("al"); break; } //End If if (ends("entli")) { r("ent"); break; } //End If </pre>	<pre> if (ends("eli")) { r("e"); break; } //End If if (ends("ousli")) { r("ous"); break; } //End If break; case 'o': if (ends("ization")) { r("ize"); break; } //End If if (ends("ation")) { r("ate"); break; } //End If if (ends("ator")) { r("ate"); break; } //End If break; case 's': if (ends("alism")) { r("al"); break; } //End If if (ends("iveness")) { r("ive"); break; } //End If if (ends("fulness")) { r("ful"); break; } //End If if (ends("ousness")) { r("ous"); break; } //End If break; </pre>
<pre> case 't': if (ends("aliti")) { r("al"); break; } //End If if (ends("iviti")) { r("ive"); break; } //End If if (ends("biliti")) { r("ble"); break; } //End If break; case 'g': if (ends("logi")) { r("log"); break; } //End If } // End switch (b[k-1]) } //End Step3() //***** </pre>	

Figure 7.30 step3 in stemming process

```

/*****
  /* step4() deals with -ic-, -full, -ness etc. similar strategy to step3. */
  private final void step4()
  {
    switch (b[k])
    {
      case 'e':
        if (ends("icate"))
        {
          r("ic"); break;
        } //End If
        if (ends("ative"))
        {
          r(""); break;
        } //End If
        if (ends("alize"))
        {
          r("al"); break;
        } //End If
        break;
      case 'i':
        if (ends("iciti"))
        {
          r("ic"); break;
        } //End If
        break;
      case 'l':
        if (ends("ical"))
        {
          r("ic"); break;
        } //End If
        if (ends("ful"))
        {
          r(""); break;
        } //End If
        break;
      case 's':
        if (ends("ness"))
        {
          r(""); break;
        } //End If
        break;
    } //End switch (b[k]) Cases
  } //End Step4()
*****/

```

Figure 7.31 step4 in stemming process

```

/*****
/* step5() takes off -ant, -ence etc., in context <c>vcvc<v>. */
private final void step5()
{
  if (k == 0) return; /* for Bug 1 */
  switch (b[k-1])
  {
    case 'a':
      if (ends("al")) break;
      return;
    case 'c':
      if (ends("ance")) break;
      if (ends("ence")) break;
      return;
    case 'e':
      if (ends("er")) break;
      return;
    case 'i':
      if (ends("ic")) break;
      return;
    case 'l':
      if (ends("able")) break;
      if (ends("ible")) break;
      return;
    case 'n':
      if (ends("ant")) break;
      if (ends("ement")) break;
      if (ends("ment")) break; /* element etc. not stripped before the m */
      if (ends("ent")) break;
      return;
    case 'o':
      if (ends("ion") && j >= 0 && (b[j] == 's' || b[j] == 't')) break; /* j >= 0 fixes Bug 2 */
      if (ends("ou")) break;
      return; /* takes care of -ous */
    case 's':
      if (ends("ism")) break;
      return;
    case 't':
      if (ends("ate")) break;
      if (ends("iti")) break;
      return;
    case 'u':
      if (ends("ous")) break;
      return;
    case 'v':
      if (ends("ive")) break;
      return;
    case 'z':
      if (ends("ize")) break;
      return;
    default:
      return;
  } // End switch (b[k-1]) Cases
  if (m() > 1) k = j;
} //End Step5()
/*****

```

Figure 7.32 step5 in stemming process

```
/** *****  
/* step6() removes a final -e if m() > 1. */  
private final void step6()  
{  
    j = k;  
    if (b[k] == 'e')  
    {  
        int a = m();  
        if (a > 1 || a == 1 && !cvc(k-1)) k--;  
    }//End if (b[k] == 'e')  
    if (b[k] == 'l' && doublec(k) && m() > 1) k--;  
}// End step6()  
/** *****
```

Figure 7.33 step6 in stemming process

```
/** *****  
/* Stem the word placed into the Stemmer buffer through calls to add().  
* Returns true if the stemming process resulted in a word different  
* from the input. You can retrieve the result with  
* getResultLength()/getResultBuffer() or toString().*/  
public void stem()  
{  
    k = i - 1;  
    if (k > 1)  
    {  
        step1(); step2(); step3(); step4(); step5(); step6();  
    }  
    i_end = k+1; i = 0;  
}//End stem()  
/** *****  
}// End class Stemmer
```

Figure 7.34 stem() method in stemmer class

References & Appendix

```

#include <math.h>
#include "mex.h"
#include "matrix.h"

unsigned int numnonzeros(const mxArray*);

/*
mex_EMstep performs one step of (T)EM given the parameters
usage: Y = mex_EMstep(X,C,Pw_z,Pz_d)
or      Y = mex_EMstep(X,C,Pw_z,Pz_d,beta)

where 'X' is the term-document matrix, 'C' the normalization
constant (evaluated at the non zero points of 'X', 'Pw_z' the
conditional distribution over words given the topics, 'Pz_d'
the document conditioned distribution over the topics.
'beta' \leq 1 for tempered EM. (default: 1)
'X' and 'C' have to be sparse (and of the same structure)

Peter Gehler
Max Planck Institute for biological Cybernetics
pgehler@tuebingen.mpg.de
Feb 2006
*/

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[]) {
    double *XPz_dw, *X, *Pw_z, *Pz_d, *C, sumXPz_dw;
    double *Pz_d_out, *Pw_z_out, *sumX;
    double beta;
    mxArray *p;
    int *ir_x, *ir_xy, *jc_x, *jc_xy;
    unsigned int indc, indx, next_indc, element_count, nn, ww;
    unsigned int nWords, nTopics, nDocs;
    bool temperedEM;

    if (!!(mxiSparse(prhs[0])) || !(mxiSparse(prhs[1])))
        || ((nlhs==2) || ((nrhs==4) || (nrhs==5))) {
        printf("usage: Y = mex_EMstep(X,C,Pw_z,Pz_d)\n");
        printf(" or: Y = mex_EMstep(X,C,Pw_z,Pz_d,beta)\n");
        printf(" where 'X' is the data matrix and 'C' the normalizing constant\n");
        printf(" both have to be sparse\n");
        printf(" 'beta' is the temperature parameter for TEM or 1 if not given\n");
        printf(" this function performs one step of EM and returns the updated\n");
        printf(" parameters. \n");
        return;
    }

    nWords = mxGetM(prhs[0]);
    nDocs = mxGetN(prhs[0]);

    nTopics = mxGetN(prhs[2]);

    jc_x = mxGetJc(prhs[0]);
    ir_x = mxGetIr(prhs[0]);
    X = mxGetPr(prhs[0]);

    C = mxGetPr(prhs[1]);

    Pw_z = mxGetPr(prhs[2]);
    Pz_d = mxGetPr(prhs[3]);

    if (nrhs == 5)
        beta = *mxGetPr(prhs[4]);
    else
        beta = 1;

    temperedEM = beta != 1;

    /* create output matrices */
    plhs[0] = mxCreateDoubleMatrix(nWords, nTopics, mxREAL);
    Pw_z_out = mxGetPr(plhs[0]);

    plhs[1] = mxCreateDoubleMatrix(nTopics, nDocs, mxREAL);
    Pz_d_out = mxGetPr(plhs[1]);

    p = mxCreateSparse(nWords, nDocs, numnonzeros(prhs[0]), mxREAL);
    jc_xy = mxGetJc(p);
    ir_xy = mxGetIr(p);
    XPz_dw = mxGetPr(p);

    sumX = malloc(nDocs * sizeof(double));
    /*
    p = mxCreateDoubleMatrix(nDocs, 1, mxREAL);
    sumX = mxGetPr(p);
    */
    if (!(temperedEM)) { /* plain EM */
        for (nn=0; nn<nTopics; nn++) { /* for every topic */
            sumXPz_dw = 0;
            element_count = 0;
            jc_xy[0] = 0;
            indx = 0;

            for (indc = 0; indc < nDocs; indc++) { /* loop over documents */
                sumX[indc] = 0;
                jc_xy[indc+1] = jc_xy[indc];
                next_indc = jc_x[indc+1]; /* next column index */
                while (indx < next_indc) {
                    jc_xy[indc+1]++;
                    /* copy element of sparse matrix */
                    ir_xy[element_count] = ir_x[indx];
                    XPz_dw[element_count] = X[indx] * Pz_d[nn+nTopics*indc] *
                    Pw_z[nn*nWords+ir_xy[element_count]] / C[indx];
                    /* this works for one topic */
                    /* XPz_dw[element_count] = X[indx] * Pz_d[indc] *
                    Pw_z[ir_xy[element_count]] / C[indx]; */
                    /* this check might be deleted at some point */
                    if (mxIsNaN(XPz_dw[element_count])) {
                        printf("isnan!\n");
                        XPz_dw[element_count] = 0;
                    }
                    else
                        sumXPz_dw += XPz_dw[element_count];

                    /* calculate the total sum of the elements in X */
                    sumX[indc] += X[indx];

                    /* update sum over rows and sum over columns */
                    Pw_z_out[nn*nWords+ir_xy[element_count]] += XPz_dw[element_count];
                    Pz_d_out[nn+nTopics*indc] += XPz_dw[element_count];

                    indx++;
                    element_count++;
                }
            }
            /* normalize Pw_z and Pz_d */
            for (ww=0; ww<nWords; ww++)
                Pw_z_out[nn*nWords+ww] /= sumXPz_dw;
            for (ww=0; ww<nDocs; ww++)
                Pz_d_out[nn+nTopics*ww] /= sumX[ww];
        }
    }
    else { /* tempered EM version - see comments from above */
        for (nn=0; nn<nTopics; nn++) {
            sumXPz_dw = 0;
            element_count = 0;
            jc_xy[0] = 0;
            indx = 0;
            for (indc = 0; indc < nDocs; indc++) {
                sumX[indc] = 0;
                jc_xy[indc+1] = jc_xy[indc];
                next_indc = jc_x[indc+1]; /* next column index */
                while (indx < next_indc) {
                    jc_xy[indc+1]++;
                    ir_xy[element_count] = ir_x[indx];
                    XPz_dw[element_count] = X[indx] * pow(Pz_d[nn+nTopics*indc] *
                    Pw_z[nn*nWords+ir_xy[element_count]], beta) / C[indx];
                    if (mxIsNaN(XPz_dw[element_count]))
                        XPz_dw[element_count] = 0;
                    else
                        sumXPz_dw += XPz_dw[element_count];
                    sumX[indc] += X[indx];
                    Pw_z_out[nn*nWords+ir_xy[element_count]] += XPz_dw[element_count];
                    Pz_d_out[nn+nTopics*indc] += XPz_dw[element_count];
                    indx++;
                    element_count++;
                }
            }
            for (ww=0; ww<nWords; ww++)
                Pw_z_out[nn*nWords+ww] /= sumXPz_dw;
            for (ww=0; ww<nDocs; ww++)
                Pz_d_out[nn+nTopics*ww] /= sumX[ww];
        }
    }
    free(sumX);
    mxDestroyArray(p);
    return;
}

/* count number of nonzero entries */
unsigned int numnonzeros(const mxArray *prhs) {
    unsigned int nnz;

    if (mxiSparse(prhs))
        nnz = *mxGetJc(prhs) * mxGetN(prhs);
    else if (mxiDouble(prhs)) {
        int s=0, i, n;
        double *xr, *x1;
        nnz = 0;
        n = mxGetNumberOfElements(prhs);
        xr = mxGetPr(prhs);
        x1 = mxGetPi(prhs);
        if (x1 != NULL)
            for (i=0; i<n; i++) {
                if (xr[i] != 0) nnz++;
                else if (x1[i] != 0) nnz++;
            }
        else
            for (i=0; i<n; i++) if (xr[i] != 0) nnz++;
    }
    return (nnz);
}

```

Figure 7.35 "mex_EMstep.c" PLSI implementation.

References & Appendix

```
#include <math.h>
#include "mex.h"
#include "matrix.h"

/*
 mex_Pw_d.c

 computes the normalization constant during EM learning for PLSI.
 The elements are computed only at those positions needed, therefore

 Peter Gehler
 */

unsigned int numnonzeros(const mxArray*);

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray*prhs[] ) {
    double *Pw_d, *Pr_X, *Pz_z, *Pz_d;
    double beta;
    int *ir_x, prev_ir_x, *ir_Pw_d, *jc_x, *jc_Pw_d, *bad_index;
    int element_count, nTopics;
    int nn, ii, jj;
    unsigned int nWords, nDocs, indc, indx, indy, next_indx;
    bool temperedEH;

    /* sanity check of the parameters */
    if (!( (mxIsSparse(prhs[0])) || !( (nrhs==3) || (nrhs==4) ) ) ) {
        printf("usage : C = mex_Estep(X,Pw_z,Pz_d)\n");
        printf(" or : C = mex_Estep(X,Pw_z,Pz_d,beta)\n");
        printf(" computes the normalization constant 'C' only for those values\n");
        printf(" which are needed (nonzeros of X).
           " If no 'beta' is given this function\n");
        printf(" does plain EM, TEM otherwise\n");
        return;
    }

    nWords = mxGetM(prhs[0]);
    nDocs = mxGetN(prhs[0]);
    nTopics = mxGetM(prhs[2]);
    jc_x = mxGetJc(prhs[0]);
    ir_x = mxGetIr(prhs[0]);
    Pr_X = mxGetPr(prhs[0]);

    Pw_z = mxGetPr(prhs[1]);
    Pz_d = mxGetPr(prhs[2]);

    if (nrhs == 4)
        beta = *mxGetPr(prhs[3]);
    else
        beta = 1;

    temperedEH = beta!=1;

    /* sanity check of the parameters */
    if (!( ( mxGetM(prhs[1]) == nWords ) || !( mxGetN(prhs[1]) == nTopics ) ) ) {
        printf("Dimensions mismatch of Pw_z, should be of size %d x %d\n",
            nWords, nTopics);
        return;
    }
    if (!( ( mxGetM(prhs[2]) == nTopics ) || !( mxGetN(prhs[2]) == nDocs ) ) ) {
        printf("Dimensions mismatch of Pz_d, should be of size %d x %d\n",
            nTopics, nDocs);
        return;
    }

    /* create sparse output matrix */
    plhs[0] = mxCreateSparse(nWords, nDocs, numnonzeros(prhs[0]), mxREAL);
    jc_Pw_d = mxGetJc(plhs[0]);
    ir_Pw_d = mxGetIr(plhs[0]);
    Pw_d = mxGetPr(plhs[0]);

    element_count = 0;
    jc_Pw_d[0] = 0;
    indx = 0;

    if (!(temperedEH)) { /* plain EM */
        for (indc = 0; indc < nDocs; indc++) {
            jc_Pw_d[indc+1] = jc_Pw_d[indc];
            next_indx = jc_x[indc+1]; /* next column index */
            while (indx < next_indx) { /* while nonzero entries in the matrix */
                jc_Pw_d[indc+1]++; /* point to next index */

                /* copy element of sparse matrix */
                ir_Pw_d[element_count] = ir_x[indx];

                ii = indc;
                jj = ir_Pw_d[element_count];

                Pw_d[element_count] = 0;

                /* sum over all topics */
                for (nn=0; nn<nTopics; nn++)
                    Pw_d[element_count] += Pz_d[ii*nTopics+nn] * Pw_z[jj+nWords*nn];

                indx++;
                element_count++;
            }
        }
    } else { /* tempered EM - see comments in plain EM */
        for (indc = 0; indc < nDocs; indc++) {
            jc_Pw_d[indc+1] = jc_Pw_d[indc];
            next_indx = jc_x[indc+1];
            while (indx < next_indx) {
                jc_Pw_d[indc+1]++;
                ir_Pw_d[element_count] = ir_x[indx];

                ii = indc;
                jj = ir_Pw_d[element_count];
                Pw_d[element_count] = 0;
                for (nn=0; nn<nTopics; nn++)
                    Pw_d[element_count] += Pz_d[ii*nTopics+nn] * Pw_z[jj+nWords*nn] *
                        pow(Pz_d[ii*nTopics+nn], beta);
                indx++;
                element_count++;
            }
        }
    }

    return;
}

/* fast implementation of nnz(..) */
unsigned int numnonzeros(const mxArray*prhs)
{
    unsigned int nnz;

    if (mxIsSparse(prhs))
        nnz = *mxGetJc(prhs)+mxGetN(prhs);
    else if (mxIsDouble(prhs)) {
        int i, n;
        double *xr, *xi;
        nnz = 0;
        n=mxGetNumberOfElements(prhs);
        xr=mxGetPr(prhs);
        xi=mxGetPi(prhs);
        if (xi!=NULL)
            for (i=0; i<n; i++){
                if (xr[i]!=0) nnz++;
                else if (xi[i]!=0) nnz++;
            }
        else
            for (i=0; i<n; i++) if (xr[i]!=0) nnz++;
    } else
        mexErrMsgTxt("Function not defined for variables of input class");

    return(nnz);
}
```

Figure 7.36 "mex_Pw_d.c" PLSI implementation

```

#include "mex.h"
#include "matrix.h"

/*
logL = mex_logL(X,Pw_d,Pd)

where X is the term-document matrix, Pw_d the distribution over the words
given the documents and Pd the prior distribution over the documents.
X and Pw_d need to be sparse (and of the same structure)

Peter Gehler
Max Planck Institute for biological Cybernetics
pgehler@tuebingen.mpg.de
Feb 2006
*/

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray*prhs[]) {
    double *X, *Pw_d, *Pd, *logL;
    int *ir_x, *jc_x;
    unsigned int nDocs, indc, indx,next_indx;

    if (((nlhs!=1)|| (nrhs!=3))
        || (!(mxIsSparse(prhs[0])) || (!(mxIsSparse(prhs[1]))))) {
        printf("usage: logL = mex_logL(X,Pw_d,Pd)\n");
        printf("where 'X' and 'Pw_d' have to be sparse\n");
        return;
    }

    nDocs = mxGetN(prhs[0]);

    X = mxGetPr(prhs[0]);
    jc_x = mxGetJc(prhs[0]);
    ir_x = mxGetIr(prhs[0]);

    Pw_d = mxGetPr(prhs[1]);
    Pd = mxGetPr(prhs[2]);

    /* create output matrix */
    plhs[0] = mxCreateDoubleMatrix(1, 1, mxREAL);
    logL = mxGetPr(plhs[0]);
    logL[0] = 0;

    /* loop over only nonzero entries of the data matrix */
    indx = 0;
    for (indc = 0; indc < nDocs; indc++) {
        next_indx = jc_x[indc+1]; /* next column index */
        while (indx < next_indx) {
            logL[0] += X[indx] * log(Pd[indc] * Pw_d[indx]);
            indx++;
        }
    }

    return;
}

```

Figure 7.37 "mex_logL.c" PLSI implementation

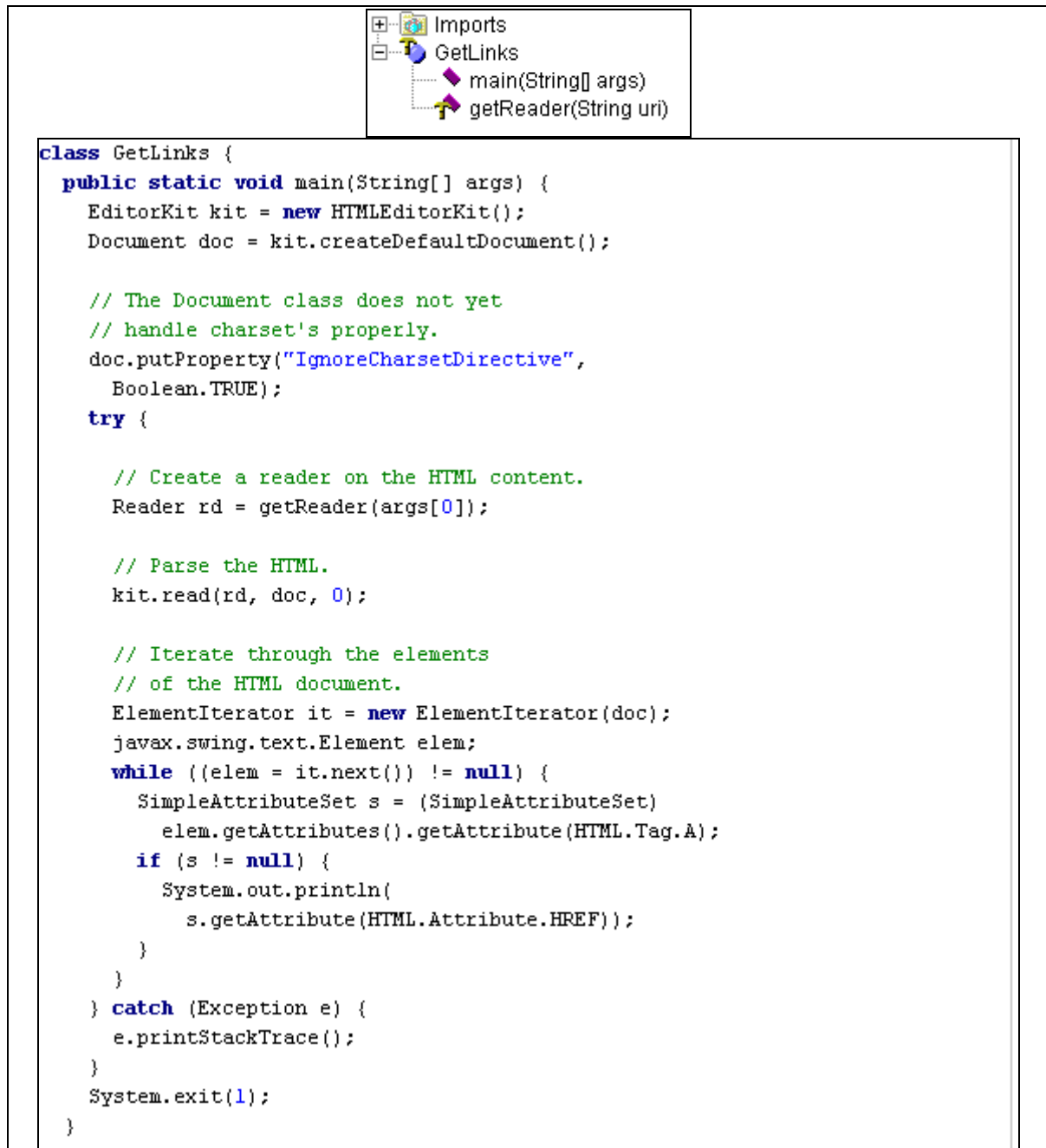


Figure 7.38 Getlinks class


```
// Returns a reader on the HTML data. If 'uri' begins
// with "http:", it's treated as a URL; otherwise,
// it's assumed to be a local filename.
static Reader getReader(String uri)
    throws IOException {
    if (uri.startsWith("http:")) {

// Retrieve from Internet.
        URLConnection conn =
            new URL(uri).openConnection();
        return new
            InputStreamReader(conn.getInputStream());
    } else {

// Retrieve from file.
        return new FileReader(uri);
    }
}
}
```

Figure 7.39 Link Recognizer

Intelligent Focused Information Agent

Dr. Rushdi A. Hamamreh

rushbeth@hotmail.com

Al Quds University, Jerusalem Palestine,
Higher educational collage

A.Mohsen K. Qawasmih

aqawasmi@eng.alquds.edu

ABSTRACT

This paper describes techniques for developing distributed and adaptive agent that coordinate to retrieve, filter and recommend information relevant to the owner, from various web sources.

The knowledge of agent based on semantic indexing by analyzing multiple topics in HTML pages.

General Terms/Topics/Keywords

Agent, Information Retrieval, Topic-Analyzer, Latent semantic indexing, Adaptation, AUML.

1. Introduction

A continuous growth of the internet usage with billions of published documents and data distributed around the world, demands a useful, fast, accurate and intelligent search system that satisfies users needs and queries. We noticed From IWS (Internet world stats) website, the rapid growth specially the last few years. We can see on Mar-2007 that the internet population grows up to 6,574,666,417 while the internet usage latest data was about 1,114,274,426. and this indicates the huge amount of document that is distributed around the world [8,1].

It's well known that search engines with centralized architecture can't index the whole Internet because the exponential growth of the number of documents published in the Internet. Search engine with distributed architecture is scalable solution of this problem [3].

The study of multi-agent systems began in the field of distributed artificial intelligence (DAI) about 20 years ago.

Our system is based on Intelligent Information Agents aims at helping the user and melting together the multi-agent system (MAS) and the information access technologies by investigating to what extent methods from Artificial Intelligence, Database Systems and Information Retrieval (IR) can be applied to information discovery by themes of information agents in the Internet and the World Wide Web.

In the framework of our suggested architecture, we use a set of topic target collections of electronic documents published in the Internet. These collections belong to different owners who are responsible for their content, indexing and quality of search. Administrator's demand is automatically propagated to one or more collections with topics relevant to his target topic[1,2,4].

2. Why .. Agent?

An agent is a computational entity such as a software program or a robot that can be viewed as perceiving and acting upon its environment and that is autonomous in that its behavior at least partially depends on its own experience. As an intelligent entity, an agent operates flexibly and rationally in a variety of environmental circumstances given its perceptual and effectual equipment. Behavioral flexibility and rationality are achieved by an agent on the basis of key processes such as problem solving, planning, decision making, and learning.[11,2]

Today these systems are not simply a research topic, but are also beginning to become an important subject of academic

teaching and industrial and commercial application.[5,7]

Our agent is expected to establish new cooperation among research groups in the related areas mentioned above, but also to strengthen existing contacts and focus scattered efforts for research on and development of intelligent information agents.

In particular, managing and controlling such networks, the services they provide, and the communications they involve, is crucial to keep Internet a useful tool in the future. However, there is a growing awareness that current centralized IR architectures will soon reach the limits of their scalability. We argue that distributed but coordinated mechanisms that support adaptation and self-optimization of Information Agent societies can be an answer to this problem.[8,4]

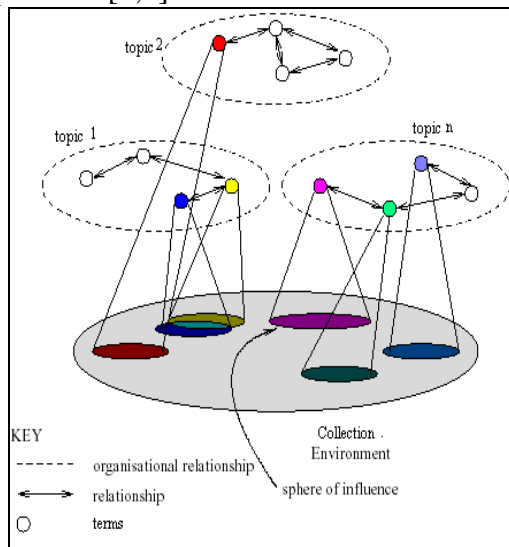


Figure 1 system environment.(Shoham 1996-modified)

3. Agent Architecture

In a distributed agent framework, we conceptualize a dynamic community

of agents, where multiple agents contribute services to the community. When external services or information are required by a given agent, instead of calling a known subroutine or asking a specific agent to perform a task, the agent submits a high-level expression describing the needs and attributes of the request to a specialized Facilitator agent. The Facilitator agent will make decisions about which agents are available and capable of handling sub-parts of the request, and will manage all agent interactions required to handle the complex query.

The advantage of such distributed agent architecture allows the construction of systems that are more flexible and adaptable than distributed object frameworks. Individual agents can be dynamically added to the community, extending the functionality that the agent community can provide as a whole. The agent - system is also able to adapt to available resources in a way that hard-coded distributed objects systems can't.

Using AUML (Agent UML) we will capture the MAS complexity by role decomposition and controls MAS environment dynamicity by role/agent entities separation. In terms of modeling, AUML supports the idea of UML extension toward Agent UML, which results to the integration of agent classes, role classes and interaction protocols to UML.[10,2]

3.1 Role A (Document Fetcher):-

This Agent Role uses wget utility for document-downloading from the internet. The link of this document is

contains a queue of links to be fetched. Links queue starts from a set of start Links presented by the administrator.

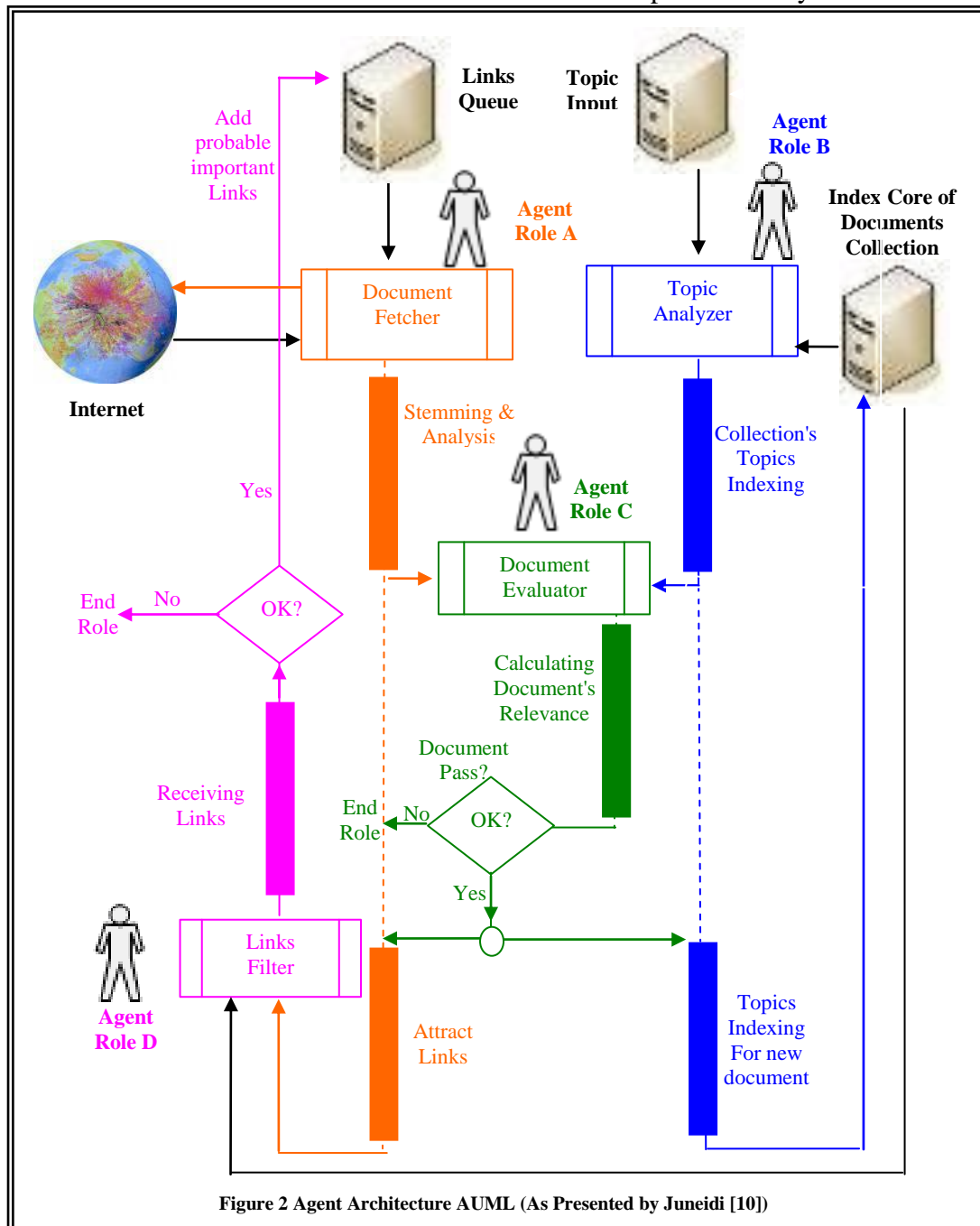


Figure 2 Agent Architecture AUML (As Presented by Juneidi [10])

taken from a storage volume which Link from this queue is assigned estimation of usefulness of this Link for seeking of new relevant documents. At the first step the newly included to this queue Link is assigned number 1 as its usefulness.

The next stage of this role is the stemming stage. Logical view of documents from full text to a set of

Every indexed terms. This stage includes Accent spacing, Noun grouping, Stop words removing ... until reaching index terms from a full text.

After that, the index terms of the fetched document will be handed to Agent role C, which is responsible to figure out if the document is relevant or not.

If the document is relevant, Agent role A starts to extract all links from this document because the probability of relevance of these links is high. These links is handed directly to Agent Role D.

3.2 Role B (Topic Analyzer):-

Using PLSI arithmetic method, this Role is responsible for two major stages:-

Stage 1, includes receiving main target topic terms that is produced by the Administrator and stored in the topic input volume. PLSI method is used to give a weight these terms, in addition to index topic terms that comes from relevant documents stored in the Index core Documents collection. Target topics terms weights are continuously modified when each new relevant document is added to the collection, at the same time these modifications are saved in Topic Input storage volume, and this loop increases the smartness of the agent. These modifications are also handed to Agent Role C, which is responsible to figure out if the document is relevant or not.

Stage 2, starts when Agent Role C decides that the fetched document is relevant, it starts to analyze the topics of document index terms using PLSI before adding the relevant document to the collection.

3.3 Role C (Document Evaluator):-

This role is responsible to receive index terms from Role A, and Topic index from Role B, and starts to calculate the relevance of the document, and the weight of it. And then determine comparing with a target threshold, whether the document is relevant or not. If the result is positive, then both of roles A and B start their mission on this new gift. But for negative result Role C ends its job on this not useful document.

3.4 Role D (Links Filter):-

At every next step Role A chose from queue a Link with maximum value

of estimation of its usefulness, downloads it and evaluates it. If this document is accepted by evaluator then at next steps agent randomly chose links presented in its text and includes them into Links queue with usefulness estimation equal 1. If a downloaded document isn't accepted by evaluator, then estimation of usefulness of a Link of a document, where link to this document occurs, is decreased. As a result, estimation of the Link usefulness is approximation of probability of relevance of a link from the document to the collection topic.

This role has to make sure that all attracted links are useful and not repeated (already checked before) in order to increase performance. And this is done with help of the stored documents collection description. So every link has to be filtered and the role decides whether to add it to Queue or not (which means to end role).

4.1 PLSI Method:-

Using PLSI arithmetic method, this Role B and C is responsible to analyze the whole set of documents from this collection and create the collection description which reflects the main subjects presented in this collection. We've used for this propose probabilistic latent semantic indexing [3,5].

The goal of the latent semantic indexing is extraction of latent factors which reflect a set of narrow topics presented in the given collection.

Let $\mathbf{z} \in \mathbf{Z} = \{\mathbf{z}_1, \dots, \mathbf{z}_k\}$ be set of these factors.

Let denote

- $P(\mathbf{z}_i)$ – probability that randomly selected document from the collection best of all corresponds to the topic \mathbf{z}_i
- $P(\mathbf{d}|\mathbf{z})$ – probability that for the given factor \mathbf{z}_i this factor best of all corresponds to the document \mathbf{d}_i
- $P(\mathbf{w}|\mathbf{z})$ – probability that for the given factor \mathbf{z}_i this factor best of all corresponds to the word \mathbf{w}_j .

Here $\mathbf{d} \in \mathbf{D} = \{\mathbf{d}_1, \dots, \mathbf{d}_N\}$ is set of all documents from the collection and $\mathbf{w} \in \mathbf{W} = \{\mathbf{w}_1, \dots, \mathbf{w}_M\}$ is set of all words from this collection.

Functions $P(\mathbf{z}_i)$, $P(\mathbf{d}|\mathbf{z})$ and $P(\mathbf{w}|\mathbf{z})$ can be estimated in the process of a likelihood function maximization. This function is presented in the following form

$$L = \sum_d \sum_w tf(d, w) \log(P(d, w)),$$

Standard Expectation Maximization algorithm is used for maximization of this function. Two steps are executed at every iteration of this algorithm. The first one is Estimation

$$P(z|d, w) = \frac{P(z)P(d|z)P(w|z)}{\sum_{z'} P(z')P(d|z')P(w|z')}.$$

The second one is Maximization

$$P(w|z) = \frac{\sum_d n(d, w)P(z|d, w)}{\sum_{d, w'} n(d, w')P(z|d, w')}$$

$$P(d|z) = \frac{\sum_w n(d, w)P(z|d, w)}{\sum_{d', w} n(d', w)P(z|d', w)}$$

$$P(z) = \frac{1}{R} \sum_{d, w} n(d, w)P(z|d, w), \quad R \equiv \sum_{d, w} n(d, w)$$

To generate the collection filter we've selected the most heavy words from \mathbf{W} . Weight of the word w is calculated as

$$\text{weight}(w) = \sum_{z \in Z} P(z)P(w|z)$$

4.2 How it works?

The goal of using PLSI method is to analyze the whole set of Administrator's queries which reflects information need of him. This analysis can be used to find new subjects which are interested to him but poorly presented in the collection core.

In order to do so we've used the following approach. At first graph of all words used in the Administrator's terms was created. Every word was presented as a vertex of this graph. Two vertices are joined with an edge if and only if the pair of corresponded words occurs in the same query. Every vertex should have a

weight which reflects the role of this word in the collection subject. Some of these words are presented in the collection core and we can use probabilistic latent semantic indexing to calculate their weights. But a part of words presented in the queries can be new (not presented in the collection core). To estimate their weights we've used the following method.

We suppose that weight of every new word should be equal to the average value of weights of words which are neighbors of this word. We've used iteration algorithm to estimate weights of all new words according to this proposal. All information about queries words and their weights is stored as queries statistics.

5. Results:-

To test our agent we've used it to generate a collection on the information retrieval topic. The collection core contains about 50 documents which were selected by a topic administrator (Security topic) relevant documents. These documents were used to generate the collection topic analyzer. We've used a set of terms (about 25) as topic-terms archive. These terms were supplied by an administrator also and were used for topic-analyzer generation. Agent starts with a set of start Links. This set contains about 5 links on relevant html documents and was presented by the administrator.

The goal of the test was to find best values for thresholds for document evaluator and topic-Analyzer. Result of the test shows that in the best case the precision of the agent is 0.722. This means that our agent downloaded a set of new documents, about 72 % of which were recommended by this agent to the collection, and our topic-administrator estimate that almost all recommended documents are relevant to the collection topic.

6. References:-

1. Lan Huang , Survey On Web Information Retrieval Technologies, Computer Science Department, State University of New York at Stony Brook, 2000.
2. Junghoo Cho and Lawrence Page, Efficient crawling through url ordering, Proceedings of the 8th International WWW conference, Canada, Tronoto, May 1999, <http://www7.scu.edu.au/programme/fullpapers/1919/com1919.htm>.
3. Rushdi Hamamreh. Multi-agent system of a distributed environment to build thematic collections // Web Instruments, № 9, 2001.
4. S. Haverkamp, Intelligent information Agents, JASIS, V49, N4, 1998.
5. Hofmann T., Latent class models for collaborative filtering. In Proceedings of the 16th International Joint conference on Artificial Intelligent, 1999.
6. Michael Wooldridge, An introduction to Multiagent Systems, John Wiley & sons Ltd,2002.
7. Multiagent Systems A Modern Approach to Distributed Modern Approach to Artificial Intelligence edited by Gerhard Weiss The MIT Press Cambridge, Massachusetts London, England.
8. Towards Agent Societies for Information Retrieval*† Holger Billhardt and Sascha Ossowski Universidad Rey Juan Carlos Dpt. of Computer Science
9. WORLD INTERNET USAGE AND POPULATION STATISTICS, IWS (Internet world stats) website www.internetworldstats.com.
10. Agent Role Locking Theory ARL, Salaheddin J. Juneidi. MIT-Press Cambridge, USA, 1-3 Nov. 2005.
11. Toward Programming paradigms for AGENT ORIENTED SOFTWARE ENGINEERING. IASTED SE Austria. Salaheddin J. Juneidi. 18-20 feb. 2004