Automated Construction Of Domain Ontologies From Lecture Notes

M.Tech Project Dissertation

Submitted in partial fulfillment of the requirements for the degree of

Master of Technology
by

Neelamadhav Gantayat
Roll No : 09305045

under the guidance of

Prof. Sridhar Iyer

Department of Computer Science and Engineering
Indian Institute of Technology, Bombay

June, 2011

Declaration

I declare that this written submission represents my ideas in my own words and where others ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Neelamadhav Gantayat
(09305045)

Date: 28th **June, 2011**

Acknowledgement

I would like to express my deep gratitude for my guide ***Prof. Sridhar Iyer***, who has always been making things simple to understand. Without his deep insight into this domain and his valuable time for this project, it would not have been possible for me to move ahead properly. He has been remarkable in his attempt to keep me motivated in this project and has always tried to improve me with proper feedback.

I would like to thank my friend ***Sagar Kale*** for his help in checking some sections for grammatical errors.

I would like to thank ***Ramkumar Rajendran*** and ***Souman Mandal***, for their constant feedback and motivation.

I would like to thank each and every one who helped me throughout my work.

Neelamadhav Gantayat
(09305045)

Abstract

Nowadays e-learning has become popular, especially with the availability of large course-ware repositories such as MIT's OCW, NPTEL and CDEEP. A variety of searching techniques, e-learning tools and systems are also available. Courseware repositories contain large amounts of lecture videos and text. When searching for lecture material on a given topic, it would be useful if the repository also indicates the topics that are pre-requisites. However, suppose a user wants to learn about a particular topic of a subject, the search tools typically return a large number of links to the user in response to his/her query (topic). Many of these are not directly related to the topic. Some of them are more advanced topics and some other links contains some irrelevant data which is nothing to do with the desired topic, so the user does not know which links to follow in order to enhance his knowledge.

In this paper we present a technique that automatically constructs the ontology (dependency graph) from the given lecture notes. We show how this ontology can be used to identify the pre-requisites and follow-up modules for a given query (lecture topic). We also provide the user with a dependency graph which gives a conceptual view of the domain. Our system extracts the concepts using "term frequency inverse document frequency (tf-idf) weighting scheme" and then determines the associations among concepts using "apriori algorithm". We have evaluated our system by comparing its results with the dependencies determined by an expert in the subject area.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Courseware repositories, such as OCW[1] and NPTEL[2], contain large amounts of data in the form of videos and text. A fine-grain (topic-level) search facility and automatic identification of pre-requisites and follow-ups for a given topic is desirable and would be useful to students. Such a feature (identification of pre-requisites of a given topic) is not available in these repositories. This feature could be built by manual tagging of the contents, but it is cumbersome to do so.

In this paper we present a technique that automatically constructs the ontology (dependency graph) from given lecture notes. We show how this ontology can be used to identify the pre-requisites and follow-up modules for a given query (lecture topic). In domain ontology, relationships between different concepts of a domain are identified. In our case, a concept corresponds to a lecture module and a relationship corresponds to whether it is a prerequisite or a follow-up of the topic. We also provide the user with a dependency graph which corresponds to a concept map and gives a conceptual view of the domain. People can often grasp ideas much more quickly by looking into the graphical representation than by reading them in a book[Cmap08]. To the best of our knowledge, there is no such system to automatically determine dependencies of topics from a repository of lecture notes.

Our system extracts the concepts using "term frequency inverse document frequency (tf-idf) weighting scheme" and then determines the associations among concepts using "apriori algorithm". We have evaluated our system by comparing its results with the dependencies determined by an expert in the subject area.

1.1 Abbreviations and acronyms

- **Ontology:** Large number of ideas and concepts to gather in a hierarchical order.
- **Query, Learning-module or concept:** Any topic related to a particular subject.
- **Most Relevant:** The PDF file which contains the topic that we are searching.

[1]http://ocw.mit.edu
[2]http://nptel.iitm.ac.in

- **Prerequisite:** Prior information which is needed before proceeding with the topic.
- **Follow-up:** Information that can be read after finishing the topic.
- **Stemming:** Finding the root (base) form of a word.
- **Name entity identification:** Finding the proper nouns, naming specific things.
- **OWL:** Web Ontology Language.
- **ngram:** Groups of n written letters, n syllables, or n words.

1.2 Motivation for MTP

The course-ware repositories like NPTEL, CDEEP and MIT's OCW provides lecture notes in the form of PDF's for a wide range of courses. Some repositories provide searching only for courses, but not for topics. If we search for any topic though it is present in a course, searching provided by these repositories cannot give the result. Most of the current search engines and search techniques available in courseware repositories use only keyword based searching, which will produce some PDF's which contains the keyword but not related to the topic.

More over current search techniques do not provide us with the prerequisites and follow-ups for a given topic. Suppose user want to learn about a particular topic, the search tool returns a large number of links to the user in response to his/her query. Instead the search tools should provide the PDF file which contains learning module for his query, and the prerequisites and follow up modules that can be learned. To achieve this objective we use Domain Ontology to create a dependency graph which will have the relation between concepts in a particular domain.

As a user we tried to search topics like Threads, TCP/IP, Ethernet etc., in some repositories. Although these topics were covered in those repositories, it showed that the topic was not available there. And in some other repositories, most of the times the search results consisted more advanced topics before the topic we searched. A detailed survey of repositories is given in the next section. Hence, an effective search facility has to be provided so that the user can get desired topics along with some related topics. Related topics can either be pre-requisites or follow-ups. Detailed repository survey is described in the next chapter.

1.3 Goal of MTP

Given a set of lecture files (PDF or Text) for a particular subject from a course-ware repository, or a Text book (soft copy), Our aim is to come up with a system which will provide the user with correct reference (link for the PDF file in case of lecture files or chapter in case of the book.) for the desired topic of the given subject. We also show a dependency graph so that the user can refer to the previous and advanced topics as required. Dependency graph will provide conceptual view of the subject to the user. We do not assume any ordering of the files or the concepts.

The system will also provide the pre-requisites and follow-ups for the topic. User can review the pre-requisite before starting the module or can refer the pre-requisites in case of any difficulties in understanding the topic. At the same time user can also refer to the follow-up to enhance his/her knowledge about the topic.

We have divided our system into three modules:

- Providing user with the link to the PDF file which consists the learning module for the query (Keyword).
- Creating a dependency graph for the entire course so that the user can have a conceptual view of the course.
- Suggestion of previous and advanced topics as required.

1.4 Solution Approach

Our technique is to extract the topics (keywords) from the given PDF files using "term frequency inverse document frequency (tf-idf) weighting scheme". Then we determine the associations among different concepts (topics) using "apriori algorithm". Then we arrange the relations in a hierarchical order. For any user query, our system provides the link for the topic, and two topics above it as pre-requisites and two topics below it as follow-ups, from the hierarchy of the ontology.

Given the contents of a course from a repository (NPTEL in our case), we do the following:

- We indexed the given text using *Lucene*, the index is used for searching, and also for finding the dependencies between different concepts.
- We used *Tf-idf weighting* to find out the important concepts and *apriori algorithm* to find out the relation between the different concepts.
- We implemented and tested our system on several courses taken from NPTEL. For effective evaluation we tested the results against the dependencies determined by an expert in the subject area.

1.5 Organization of the report

In Chapter 2, we explained the background required by a reader in order to proceed with the report and also different course-ware repositories like CDEEP, NPTEL, and MIT's OCW, and their searching strategies. Chapter 3 contains different methodologies, techniques, tools and languages for developing ontologies. Chapter 4 describes different approaches to our system. The implementation of our systems is described in Chapter 5. Our experiments to evaluate the performance of the system are shown in Chapter 6 and conclusions in Chapter 7.

Chapter 2

Background

This chapter describes courseware repositories, gives a brief overview of domain ontology and defines dependency graph.

2.1 Ontology

Ontology Borrowed from philosophy - the study of *"The nature of being"*[1]. Ontology in information system is a large number of ideas and concepts to gather in a hierarchical order. It provides a mechanism to capture information about the objects, Classes and the relationships that hold between them in some domain. The aim of ontology is to develop knowledge representations that can be shared and reused. Guber[Grub95] defined an ontology as

"A formal explicit specification of a shared conceptualization."

In ontology classes describe concepts in the domain. A class can have subclasses that represent concepts that are more specific than the superclass. Slots describe properties of classes and instances.

2.1.1 Domain Ontology

Domain Ontology is an ontology Model which provides definitions and relationships of the concepts, major theories, principles and activities in the domain. Domain ontologies provide shared and common understanding of a specific domain. Domain ontology provides particular meaning of term as they apply to that domain. For example the word *thread* has many different meaning. An ontology about the domain of operating system would model "process threads", while an ontology about the domain of "textiles" would model *thread* with different meaning.

[1]Taken from: `http://en.wikipedia.org/wiki/Ontology`

2.1.2 Applications of Ontology

Main application areas of ontology are knowledge management, Web commerce, electronic[OIL] business and e-learning.

Knowledge Management is concerned with acquiring, maintaining, and accessing an organization's data. Nowadays organizations are distributed around the world. Ontology will help in these organizations in searching, extracting and maintaining the large number of on-line documents. Ontology will give efficient searching techniques other than keyword matching.

Web commerce is extending the exiting business models with reduced costs. Some examples where web commerce can be used is online market places and auction houses. Ontology will help the customers in finding the shops that sells the desired product with quality, quantity and reduced cost. Ontology can describe the various products and help navigate and search automatically for the required information.

Electronic Business is nothing but automation of business transactions. Ontology included eBusiness will help in automation of data exchange.

E-learning To find out the dependencies between the keywords of a topic in the repositories, and to facilitate the user with more recent and relevant data.

The key difficulties in developing ontology are: (i) extensive knowledge about a subject is required and (ii) it is time-consuming. We have automated this process, in the context of lecture notes. We use domain ontology to represent relations between topics for a given course. Here we consider only one relation, which is *"follows"*. Topic-2 *follows* Topic-1 means that Topic-1 is a pre-requisite for Topic-2 and Topic-2 is a follow-up of Topic-1. In our system we first develop the domain ontology from the given set of notes. Then we refer the node which represents the user's desired topic and also provide two of its ancestor nodes as pre-requisites and two descendants as follow-ups.

The domain ontology developed by our system is also presented to the user by a graphical representation called dependency graph.

2.2 Dependency graph

A dependency graph is a directed graph which represents dependencies of several objects towards each other.

"Given a set of objects S and a transitive relation $R = S \times S$ with $(a, b) \in R$ modeling a dependency 'a needs b evaluated first', the dependency graph is a graph $G = (S, T)$ with $T \subseteq R$ and R being the transitive closure of T."[Wikidep]

Dependency graphs are represented in hierarchical order, i.e., most general concepts are at the top of the graph and the more specific and less general concepts in lower orders. Using dependency graphs we can represent the dependencies between different concepts as shown in Figure 2.1; concepts are shown by ellipses and dependencies by arrows.

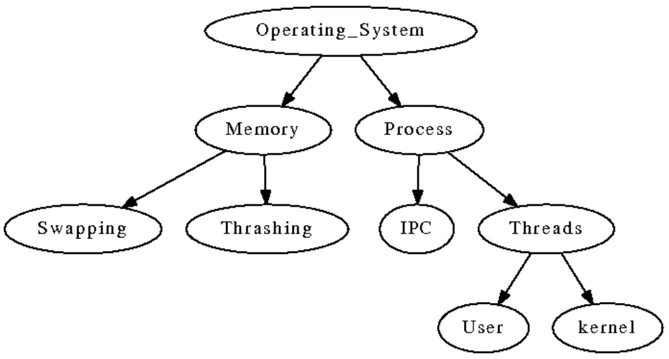


Figure 2.1: Dependency Graph for "Operating System"

A dependency graph being similar to a concept-map[Cmap08], enhances the learner's understanding of a given subject and is useful for providing summary of various interconnected and dependent topics. The key difference between a dependency graph and a concept-map is that: a concept-map can have any relation between two concepts, whereas in a dependency graph there is only one relation, that is, *depends*.

2.3 Repositories surveyed

We surveyed MIT's OCW, NPTEL, and CDEEP repositories which provide access to all of its course content for free of cost.

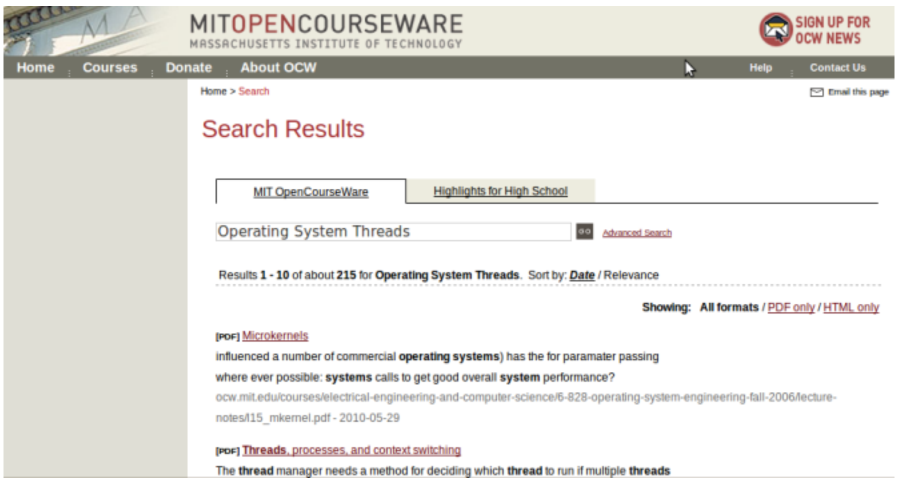
MIT's OCW

MIT OpenCourseWare is an initiative by MIT faculty to educate the students in science, technology and other areas. There are over 2,000 courses in 36 academic disciplines[2]. This content is available for download freely in the form of MIT's OpenCourseWare and there is a dedicated website `http://ocw.mit.edu/` for this. Most of the content has been made available in the form of PDF documents. Our search for the topic "Operating system Threads" gave the link for "Micro-kernels" first. It does not really help the user as more advanced topic links come before the desired links. The screen-shot is shown in Figure 2.2.

Difficulties:

- Here Micro-kernels came before the actual kernels. It will not really help the user as more advanced topic links were coming before the desired link.
- It gives some results which are not at all related to operating system threads.
- It is hard to decide from the large results, which are basic, and which are advanced topics.

[2]`http://ocw.mit.edu/about/site-statistics/monthly-reports/`



Search Results

Operating System Threads

Results **1 - 10** of about **215** for **Operating System Threads**. Sort by: ***Date*** / Relevance

Showing: All formats / PDF only / HTML only

[PDF] Microkernels
influenced a number of commercial **operating systems**) has the for paramater passing
where ever possible: **systems** calls to get good overall **system** performance?
ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-828-operating-system-engineering-fall-2006/lecture-notes/l15_mkernel.pdf - 2010-05-29

[PDF] Threads, processes, and context switching
The **thread** manager needs a method for deciding which **thread** to run if multiple **threads**

Figure 2.2: MIT's OCW search for "Operating system Threads"

National Program on Technology Enhanced Learning (NPTEL)

The National Programme on Technology Enhanced Learning (NPTEL)[3] is a project from the Ministry of Human Resource Development (MHRD), which was initiated in 1999[4]. The main idea behind NPTEL is to introduce multimedia and web technology in teaching. There are two modes of courses available. One is digital video lectures of some courses, and the other one is lectures notes in the form of PDF files. Here only course search is available, there is no topic search provided.

Difficulties:

- The search option provided works for the course names. One can't get any information about a particular query (Topic) if it does not appear in the course name.

CDEEP

The Centre for Distance Engineering Education Programme (CDEEP)[5] was started by the Indian Institute of Technology (IIT) Bombay. The main objective of CDEEP is to provide distance education in engineering and science to students outside IIT. CDEEP is offering 53 courses in 6 major areas. Different activities of CDEEP include laboratory demonstrations, transmitting classroom lectures live to the destination, develop web-based course material, tutorials, assignments, and studio recording of lectures etc. There is no search facility available in CDEEP. There is no search facility available in CDEEP. Table 2.1 shows the summary of the courseware repositories that we surveyed.

[3]http://nptel.iitm.ac.in
[4]http://nptel.iitm.ac.in/pdf/NPTEL%20Document.pdf
[5]http://www.cdeep.iitb.ac.in

Table 2.1: Comparison of different course-ware repositories

	MIT's OCW	**NPTEL**	**CDEEP**
Developers	MIT	MHRD India	IITBombay
Course Search	Yes	Yes	No
Keyword Search	Yes	No	No
Search for "Operating System Threads"	215 Results	No Result	No Result
Difficulties	More advanced Results came first.	No topic Search	No Search
Pre-Requisites/ Follow-Ups	No	No	No

2.4 Searching Tools surveyed

Google Custom Search or Google Site search

Google Custom Search or Google Site Search[6] applies the power of Google to create a customized search box for our own website. Google Site Search is a hosted search solution that enables Customize search box. It retrieves results using XML. Custom Search for our website or blog, provides fast search results.

Our search in google site search of `www.iitb.ac.in` for "Threads" returned around 409 results. None of the resultant links were related to Operating system, our domain of concern.

Google search

Google is a General purpose search engine for searching Audio, Video and text material. Our search for "operating system Threads" returned huge number of results. None of them have the link for "operating system threads" PDF of any of the repositories we surveyed. User has to try almost all the results on the first page to get into the correct link. It is very difficult for the user to search from these many options. Google does not provide the prerequisite and follow-ups for any course modules.

[6] `http://www.google.com/cse/`

Chapter 3

Literature Survey

This chapter deals with different automatic ontology generation tools, Ontology Languages and editors.

3.1 Mining based Automatic Ontology Construction[Ivan07]

Mining based techniques implement some mining techniques to retrieve the keywords from the given text documents. Mining techniques incorporate automatic key word extraction techniques in order to construct the ontology. Here the text documents can be web pages or files.

3.1.1 TERMINAE[Term99]:

The purpose of TERMINAE is to build automatic ontology from text as well as a new ontology manually. It is a computer aided Knowledge-Engineering tool written in java. TERMINAE is composed of two tools.

1. Linguistic Engineering Tool
2. Knowledge Engineering Tool

Linguistic Engineering Tool: This module allows the extraction of terminological forms (keywords) from the given corpus (Text file). Terminological forms define each meaning of a term called a notion using some linguistic relation (Parts-of-Speech) between notions such as synonyms.

Knowledge Engineering Tool: This module involves knowledge base (Ontology) management with an editor and browser for the ontology. The tool helps to represent a notion (topic or keyword) as a concept. If we want to create a new ontology then we can directly use this module which can create the ontology from scratch.

Conceptual view of TERMINAE:

- LEXTER, a term extractor, is used to extract the candidate terms (keywords) from the corpus.
- With the help of an expert, effective terms from the candidate terms are selected.
- Then conceptualize each term. That is, give definition in natural language for each notion and then translate the definition into formalism.
- Depending on the validity of the insertion we may or may not insert the concept into the ontology.
- At each step of insertion Validate the Ontology whether it serves our purpose or not.

Practical view of TERMINAE

Prerequisites[1]**:** First convert the PDF files into text using the `pdfBox`. Then use `TreeTagger` to extract the keywords and its parts-of-speech. Now process the `TreeTagger` output file with `YaTeA`, it will produce a XML file.
Process: TERMINAE assumes that the acquisition corpus has been tagged by `TreeTagger` and then processed by `YaTeA` beforehand. When we open TERMINAE it creates a folder with the project name and some sub-folders in the main folder. In the corpus folder place the corpus data and the output file of the `TreeTagger`. and in the `YaTeA` folder keep the XML file which was generated by `YaTeA`.

Now click on Linguistic level then go to `YaTeA` and then to valid occurrences/Create terminological forms, it will ask for the XML output of the `YaTeA`. Select the one that we pasted then it will ask the text file select the corpus. It will display all the unique words present in the corpus along with its frequency and List of occurrences. The main role of this term extractor results window is to allow cleaning and reorganizing the table of terms provided by `Yatea`. We can clean single word terms numbers as well as words containing some special characters according to our choice.

An expert will now select the concepts required for the creation of ontology and then for each concept go to terminological form. this module will save each concept as a XML file in the *fichesTerminologique* subfolders.

According to the users requirement the concepts which are selected in the above step may or may not be inserted into the ontology. TERMINAE can also be used individually to create a new operating system from scratch.

[1]`http://www-lipn.univ-paris13.fr/~szulman/logi/`

TERMINAE is not suitable for our system because

- Not fully automatic.
  - User should process the corpus through TreeTagger and YaTeA manually and follow the instructions.
  - An expert is required to select the most important notions(concepts) for the target ontology from the list of terms (Keywords) extracted by the tool.
  - Domain expert is also required to provide a definition of the meaning for each term in natural language.
- YaTeA will fail if there is any XML or HTML code present in the corpus text.
- Static i.e., we cannot insert new topic after creating the final ontology.

3.1.2 Ontology Development using SALT[SALT02]

It is the common idea of two different projects: The standardization of lexical and terminological resources (SALT) and the use of conceptual ontologies for information extraction and data integration (TIDIE). This approach assumes the availability of 3 types of knowledge sources

- More general and well defined ontology for the domain.
- A dictionary or any external source to discover lexical and structural relationships like WordNet.
- Consistent set of training text documents.

To extract Ontology knowledge source must

- Be of a general nature.
- Contain meaningful relations.
- Already exist in Machine readable form.
- Have a straight forward conversion into XML.

Conceptual view

The proposed architecture is given in the following figure 3.1 [SALT02].

Concept selection: Select the user required concepts from the domain. This is done by string matching between textual content and ontological data. Here two assumptions are made (1) word synonyms are considered through the use of WordNet synonym sets. (2) Multi-word terms will undergo word-level matches. For example capital-city is considered as the synonym of both capital and city.

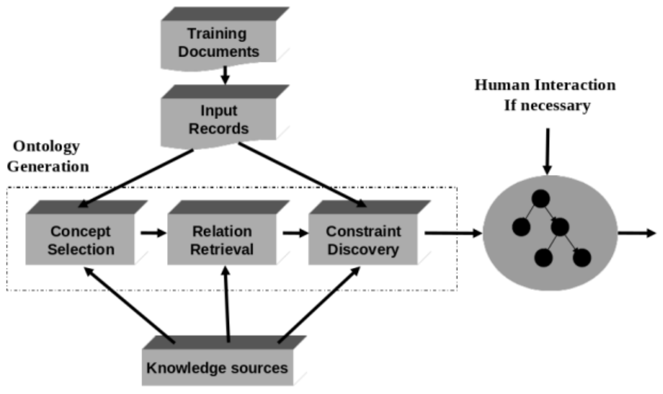


Figure 3.1: Ontology generation process

Relationship retrieval: First find out the conceptual relationships from the knowledge sources. Now construct a directed graph whose nodes are concepts. And the relationships between these concepts can be represented by paths among the concepts. To find the relationships more accurately use Dijkstra's algorithm, to find out the shortest path (more appropriate) relations among the concepts.

Constraint Discovery: constraints such as a person can have only one Date of Birth, two parents and several phone numbers follows adopted conventions.

Refining results: The output ontology may not be the final ontology which user can directly use. An expert will revise and refine the ontology.

This approach is not suitable for our system because

- It assumes more general and well defined ontology for the domain.
- It requires dictionary or any external source to discover lexical and structural relationships like WordNet.
- User intervention is required at the end of the process because it can generate more concepts then required.

3.1.3 Learning OWL ontology from free text [LIU04]

Automatic generation of ontology based on an analysis of a set of texts followed by the use of WordNet.

- First the keywords of the text are analyzed.
- These words are then searched in WordNet to find the concepts associated with these words.
- Here the Ontology generation is most automated.

This approach is not suitable for our system because

- Detail of how the terms are extracted from text is not available.
- This technique works well if there is more general reference knowledge like WordNet is available.

3.1.4 Ontology Construction for Information Selection [Khan02]

1. Terms are extracted from documents with text mining techniques.
2. Documents are grouped hierarchically according to their similarities using a modified version of SOTA algorithm.
3. Assign concepts to the tree nodes starting from leaf nodes with a method based on the Rocchio algorithm.
4. Concept assignment is based on WordNet hyponyms.
5. Bottom up approach for ontology generation.

This approach is not suitable for our system because

- It needs a more general ontology (WordNet) to define concept for the targeted ontology.

3.1.5 Comparison of Ontology construction methods

Table 3.1: Comparison of Ontology construction methods, taken from[Ivan07]

	Extraction	**Analysis**	**Generation**	**Validation**
TERMINAE	NLP tools are used, Human intervention is optional	Concept Relationship analysis (Semi-automated)	No standard Ontology representation	Purely by human
SALT	NLP Techniques fully automated	Similarity analysis of concepts	No standard Ontology representation	Limited human intervention
Learning OWL Ontology from Text	NLP Techniques, human intervention is optional,WordNet is used for keywords.	Not provided	OWL format (Human intervention optional)	Not provided
Ontology Construction for information selection	Human intervention is optional	Not provided	Human intervention optional	Not provided

Definitions:

- **Extraction:** Getting the information (concepts) needed to generate the ontology, from text documents.
- **Analysis:** Arranging the concepts in a hierarchical order.
- **Generation:** Formalizing the data i.e. generating the OWL or RDF/S file.
- **Validation:** It can be done after each step or at the end to check whether the ontology fits for our requirements or not.

3.2 Various Methods of Developing Ontology

Practically, developing an Ontology includes[NOY01]

- Defining classes in the ontology
- Arranging classes in a taxonomic (subclass-superclass) hierarchy
- Defining slots and describing allowed values for these slots.
- Filling in the values for slots for instances.

Before getting into various methods of constructing ontologies let us first emphasize on the fundamental rules in Ontology design[NOY01].

1. There is no one correct way to model a domain. There are always alternatives.
2. Ontology development is necessarily an iterative process.
3. Concepts in the ontology should be close to objects (Physical or logical) and relationship in your domain of interest. There are mostly nouns (Objects) or verbs (relationships) in sentences that describes the domain.
4. An ontology is a model of reality of the world and the concepts in the ontology must reflect this reality.

There are different methods and methodologies for developing Ontologies. Out of them we have chosen the following for study purpose. For our development we have chosen Seven-Step Method proposed by Noy and Deborah. All the methodologies more or less have the same iterative process for developing the ontology in seven-step method the steps are elaborated more and presented more clearly. We explained different methodologies in brief and Seven-Step Method in detail with an example.

3.2.1 Skeletal methodology

Proposed by Uschold and King[Grun95], Different Phases of Developing Ontology are:

1. Identifying a purpose and scope
2. Building the ontology
   (a) Ontology capture
   (b) Ontology coding
   (c) Integrating existing Ontologies
3. Evaluation
4. Documentation

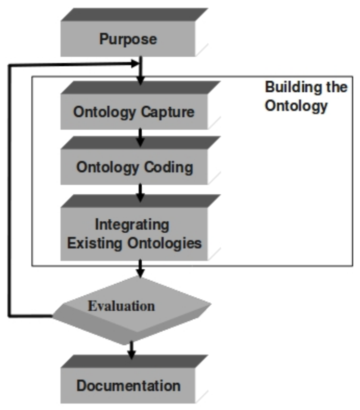


Figure 3.2: Skeletal Ontology Approach

Purpose: It is important to be clear about the purpose of ontology and the intended users of the particular ontology. Some ontologies were developed to structure a knowledge base and some other ontologies are used as a part of a knowledge base.

Building the Ontology: Ontology construction includes:

1. Capture:
   - Identification of the key concepts and relationships in the domain of interest (scoping).
   - Production of unambiguous text definitions for the concepts and relationships.
   - Identification of terms to refer to such concepts and relationships.
2. Coding: Coding is nothing but explicit representation of the conceptualization capture in the above stage in some formal language.
3. Integrating existing ontologies:
   In order to agree on ontologies that can be shared among multiple user communities, much work must be done to achieve agreement. One way forward is to make explicit all assumptions underlying the ontology.

Evaluation: Evaluation mainly deals with verification and validation that is validating the relations and verifying the purpose.

Documentation: All important assumptions should be documented, both about the main concepts defined in the ontology, as well as the primitives used to express the definitions in the ontology.

3.2.2 Practical Approach

proposed by Gavrilova[GAV05], It consists of 5-steps for creating ontology:

1. Glossary development
2. Laddering
3. Disintegration
4. Categorization
5. Refinement

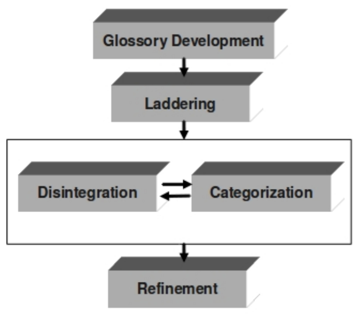


Figure 3.3: Practical Ontology Approach

Glossary development Gather all the information relevant to the described domain. The main goal of the step is selecting and verbalizing all the essential objects and concepts in the domain.

Laddering Define the main levels of abstraction. Specify the type of Ontology classification such as taxonomy, partonomy, and genealogy.

Disintegration Break high level concepts, built in the previous step, into a set of detailed ones where it is needed. This could be done via a top-down strategy trying to break the high level concept from the root of previously built hierarchy.

Categorization Detailed concepts are revealed in a structured hierarchy and the main goal at this stage is generalization via bottom-up structuring strategy. This could be done by association similar concepts to create meta-concepts from leaves of the aforementioned hierarchy.

Refinement The final step is to updating the visual structure by excluding the excessiveness, synonymy, and contradictions. As mentioned before, the main goal is harmony and clarity.

3.2.3 Knowledge Engineering Approach

It was better described in *"Development of Domain ontology for e-learning courses"*[YUN09] which was appeared in ITIME - 09 IEEE international symposium.

1. Identify purpose and requirement specification
2. Ontology acquisition
3. Ontology implementation
4. Evaluation/Check
5. Documentation

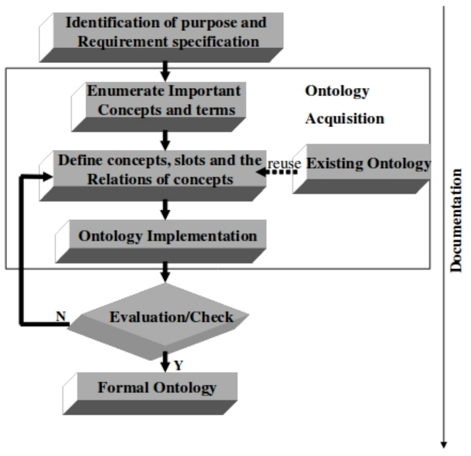

Identification of purpose and Requirement specification
Enumerate Important Concepts and terms
Ontology Acquisition
Define concepts, slots and the Relations of concepts
reuse
Existing Ontology
Ontology Implementation
N
Evaluation/Check
Y
Formal Ontology
Documentation

Figure 3.4: Knowledge Engineering Approach, Taken from [YUN09]

Identify purpose and requirement specification: Ontology purpose, scope and its intended use, i.e. the competence of the ontology.

Ontology acquisition: Capture the domain concepts based on the ontology competence. It involves

1. Enumerate important concepts and terms in this domain
2. Define concepts, properties and relations of concepts, and organize them into hierarchy structure.
3. Consider reusing existing ontology.

Ontology implementation Explicitly represent the conceptualization captured in a formal language

Evaluation/Check The ontology must be evaluated to check whether it satisfies the specification requirements.

Documentation All the ontology development must be documented, including purposes, requirements, textual descriptions of the conceptualization, and the formal ontology.

3.2.4 Seven-Step Method

It is proposed by Noy and Deborah[NOY01]. It describes the process of developing ontologies in following steps:

1. Determine the domain and scope of the ontology.
2. Consider reusing existing ontologies.
3. Enumerate important terms in the ontology.
4. Define the classes and the terms in the ontology.
5. Define the properties of classe's slots.
6. Define the faces of the slots.
7. Create instances.

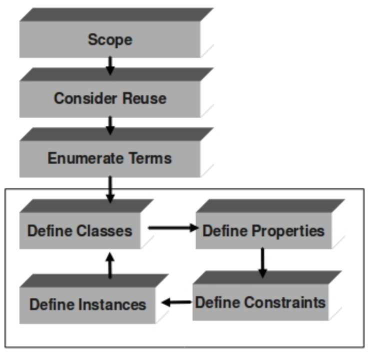


Figure 3.5: Seven-Step Ontology Approach

Determine Scope

Scope is nothing but the purpose of ontology. It should answer the following questions.

- What is domain that the ontology will cover?
- For what we are going to use the ontology?
- For what type of questions the ontology should provide answers?
- Who will use and maintain the ontology?

For Example: Let us consider that we have to develop an ontology for operating system. The purpose for development may be to find out the dependencies between the different topics of operating system.

Consider Reuse

Check if we can refine and extend existing sources for our particular domain and task. There are reusable ontologies on the web and in the literature.
Ex:

- `www.ksl.stanford.edu/software/ontolingua`
- `www.daml.org/ontologies/`
- `www.unspc.org`
- `www.roselternet.org`
- `www.dmoz.org`

For the development of our Operating System we are not able to find out the existing ontology from these repositories.

Enumerate Terms

Write down a list of all terms related to the domain that we would like to explain to a user. It is important to get comprehensive list of terms without worrying about concepts they represent relation among the terms, or any property of concepts or whether concepts or classes or slots. We can refine the terms in the subsequent steps. We used tf-idf algorithm to automatically identify the keywords.

For Example: In operating system ontology the keywords may be *Types of Computing, Types of Systems, Process Management, Memory Management, File Management etc.,*

Define Classes

There are several approaches in developing a class hierarchy

Top-down approach: This process starts with the definition of the most general concepts in the domain and subsequent specialization of the concepts.

Bottom-up approach: This type of development process starts with the definition of the most specific classes that leaves of the hierarchy with subsequent grouping of these classes into more general concepts.

Combination Approach: This is a combination of the top-down and bottom-up approaches. We define the more salient concepts first and then generalize and specialize them appropriately. We might start with a few top-level concepts and a few specific concepts. We can then relate them to a middle-level concept.

None of these three methods are better than one another. The approach to take depends strongly on the domain and the Ontology developer. The combination approach is often the easiest for many ontology developers. Whatever the approach it is we start by defining classes. From the list which is derived from Step-3 select the terms that describe objects having independent existence rather than terms that describe these objects. These terms will be classes in the ontology and will become anchors in the class hierarchy. If a class A is a superclass of class B, then every instance of B is also an instance of A, i.e., Class B represents a concept that is a “kind of” A.

For Example: If we arrange the keywords gathered above we will get an intermediate graph as shown in Figure 3.6.

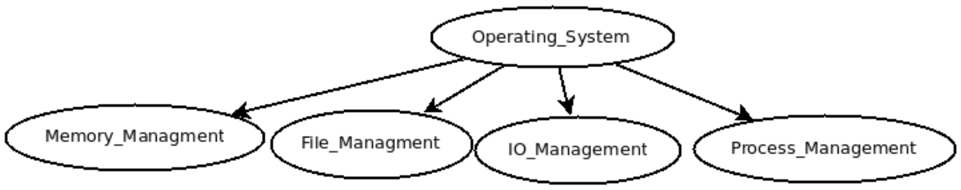


Figure 3.6: Defining classes of “Operating System”

Define Properties

Once we have defined some of the classes, we must describe internal structures of concepts. After selecting the classes from the list created by Step-3 most of the remaining terms are likely to be properties of these classes. For each property in the list we must determine which class it describes. These properties become slots attached to classes. In general, there are several types of object properties that can become slots in an ontology.

- “intrinsic” properties such as taste, and flavor of vegetables,
- “extrinsic” properties such as vegetable name and “area” it comes from.,
- Relationship to other individuals: relationship between individual member of the class and other items.

All subclasses of a class inherit the slot of that class. A slot should be attached at the most general class that can have that property.

For Example: The properties (type) of *“Thread”* are shown in the Figure 3.7.

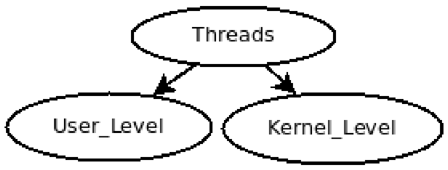


Figure 3.7: Types of “Threads”

Define Constraints

Slots can have different aspects (facets, values). The facets may describe the value type, allowed values, the number of the values (cardinality), and other features of the values the slot can take. Example:

- Value of a name slot is one string i.e. name is a slot with value type string.

Several common facets:

Slot cardinality: Slot cardinality defines how many values slot can have, for example single cardinalities (allows at most one value) and multiple cardinality (allows any number of values). Some systems allow specification of a minimum and maximum cardinality to describe the number of slot values more precisely. i.e. minimum cardinality of *'n'* means a slot must have at least *'n'* values. And similarly Maximum cardinality of *'m'* means that a slot can have at most *'m'* values.

Slot-value type: A value type facet describes what type of values can fill in the slot. Following are some of the examples of slots.

String: the value is a simple string used for slots such as name.

Number: describes slots with numeric values, more precisely can have integer and float.
Ex: price

Boolean: Slots simply Yes-No (True-False) flags.

Enumerated: It specify a list of specific allowed values for the slot
Ex: flavor slot can take strong, moderate, and delicate

Instance: These types of slots allow definition of relationships between individuals. Slots with value type instance must also define a list of allowed classes from which the instances can come.

Create instance

Define an individual instance of each class. It requires

1. Choosing a class.
2. Creating an individual instance of the class.
3. Filling in the slot values.

3.3 Ontology languages

Ontology languages are formal languages used to construct ontologies. It can formally describe the meaning of terminology used in web documents.

Need of Ontology languages Ontologies can be viewed as Database Schema but we cannot utilize the database schema. Because database schema is more rigid and can fit into set of tables whereas ontology will not fit into tables. Fensel [XMLS] points out the following differences between ontologies and schema definitions:

- A language for defining ontologies is syntactically and semantically richer than common approaches for databases.
- The information that is described by an ontology consists of semi-structured natural language texts and not tabular information.
- An ontology must be a shared and legal terminology because it is used for information sharing and exchange.

3.3.1 History of Ontology Languages [Fern03]

At the beginning of the 1990's, a set of AI-based ontology implementation languages were created. Following Figure - 3.8 describes the hierarchy of different ontology languages.

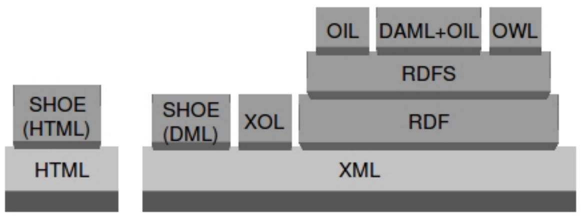


Figure 3.8: Stack of Ontology Markup Languages taken from [Fern03]

SHOE was built in 1996 as an extension of HTML, in the University of Maryland. It uses set of tags which are different form the HTML specification thus it allows insertion of ontologies in HTML documents. SHOE just allows representing concepts, their taxonomies, n-ary relations, instances and deduction rules.

Then XML was created and widely used as a standard language for exchanging information on the web. Then SHOE syntax was modified to includes XML, and some other ontology languages are also built on XML.

XOL was developed by the AI center of SRI international, in 1999. It is a very restricted language where only concepts, concept taxonomies and binary relations can be specified. No inference mechanisms are attached to it. It is mainly designed for the exchange of ontologies in the biomedical domain.

Then RDF was developed by the W3C (The world wide web consortium) as a semantic-network based language to describe Web resources. RDFSchema was built by the W3C as an extension to RDF with frame-based primitives. The combination of both RDF and RDFSchema is normally known as RDF(S). RDF(S) is not very expressive. It just allows the concepts, concept taxonomies and binary relations.

Three more languages have been developed as extensions to RDF(S): OIL, DAML + OIL and OWL. OIL was developed in the framework of the European IST project On-To-Knowledge.

It adds frame-based Knowledge Representation primitives to RDF(S), and its formal semantics is based on description logics.

DAML + OIL was created by a joint committee from the US and the EU in the context of the DARPA project DAML. DAML + OIL also adds DL-based KR primitives to RDF(S). Both OIL and DAML + OIL allow representing concepts, taxonomies, binary relations, functions and instances. Many efforts are being put to provide reasoning mechanisms for DAML + OIL.

Finally, in 2001, the W3C formed a working group called Web-Ontology (WebOnt) Working Group. The aim of this group was to make a new ontology markup language for the Semantic Web, called OWL (Web Ontology Language).
Brief Description of the Languages

3.3.2 XML (Extended Markup Language)

XML[XML] is a markup language for delivery of documents containing structured information over the web. Structured information contains both content and some indication of what role that content plays. In HTML the tag semantics and the tag set are fixed. It does not provide arbitrary structure. XML specifies neither semantics nor a tag set i.e., there is no fixed tags in XML, and XML provides a facility to define tags and the structural relationships between them. XML is created so that richly structured documents can be used over the web.

XML documents are composed of markup and content. Following kind of markups can occur in XML document: elements, comments, processing instructions etc.
Elements: elements identify the nature of the content they surround, some elements may be empty or non-empty if an element is not empty, it begins with a start-tag, <*element*> and ends with an end-tag, <\element> attributes, are name-value pairs that occur inside start-tags after the element name. For example

```
<div class="preface">
```

is a div element with the attribute class having the value preface. In XML all attribute values must be quoted.
Comments: Comments begin with <!- - and end with - ->. Comments can contain any data except the literal string - -. We can place comments between markups, anywhere in the document.
Processing Instructions: Processing instructions are an escape sequences to provide information to an application. Like comments, they are not textually part of the XML document, but the XML processor is required to pass them to an application. Processing instructions have the form:

```
<?name pidata?>
```

Table 3.2: Book Store

Book Id	Title	Author	Year	Price
059600	XML	John	2005	30
059601	Javascript	David	2003	29.99

Basic XML code looks like

For the above text the XML code is

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<bookstore>
  <book Id="059600">
    <title lang="en">XML</title>
    <author>John</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book Id="059601">
    <title lang="en">Javascript</title>
    <author>David</author>
    <year>2003</year>
    <price>29.99</price>
  </book>
</bookstore>
```

The first line is the XML declaration. It defines the XML version (1.0) and the encoding used (ISO-8859-1 = Latin-1/West European character set).

The next line describes the root element of the document (like saying: “this document is for bookstore”) i.e. <bookstore>, The next 4 lines describe 4 child elements of the root (title, author, year, price), And finally the last line defines the end of the root element <bookstore>

We can attach cascaded style sheet to the XML document by adding the following code in the second line

```
          <?xml-stylesheet type="text/css" href="book.css"?>
```

For the above data we can draw a graph for each book. the following figure shows the graphical representation for the book JavaScript

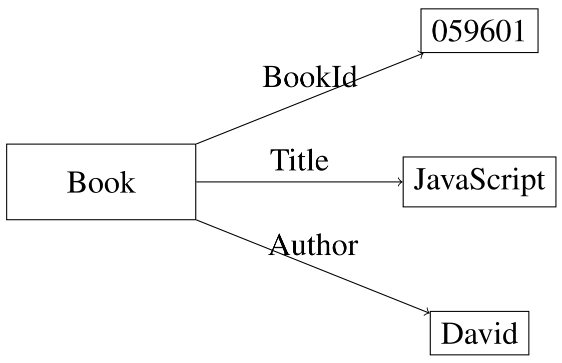


Figure 3.9: graphical representation

XMLS(XML Schema)

XML Schema[XMLS] is a means for defining constraints on well formed XML documents. It provides basic vocabulary and predefined structuring mechanisms for providing information in XML. XML Schemas are extensible, because they are written in XML.

That is we can reuse our Schema in other Schemas, we can create our own data types derived from the standard types and we can Reference multiple schemas in the same document. XML Schema provides several significant improvements:

- XML Schema definitions are themselves XML documents. The clear advantage is that all tools developed for XML (e.g., validation or rendering tools) can be immediately applied to XML schema definitions, too.
- XML Schemas provides a rich set of datatypes that can be used to define the values of elementary tags.
- XML Schemas provides a much richer means for defining nested tags (tags with subtags)
- XML Schemas provides the namespace mechanism to combine XML documents with heterogeneous vocabulary.

With XML Schemas, the sender can describe the data in a way that the receiver will understand. A date like: ”03-11-2004” will, in some countries, be interpreted as 3.November and in other countries as 11.March.

However, an XML element with a data type like this:

```
<date type="date">2004-03-11</date>
```

ensures a mutual understanding of the content, because the XML data type “date” requires the format “YYYY-MM-DD”.

It provides syntax for structured documents but no semantic constraints on the meaning of these documents.

Let us consider an example

Table 3.3: Message

To	From	Heading	Body
John	David	Reminder	Meeting is cancelled.

For the above XML code the XML Schema will be

```
 <?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.cse.iitb.ac.in"
xmlns="http://www.w3schools.com"
elementFormDefault="qualified">

<xs:element name="note">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
      <xs:element name="heading" type="xs:string"/>
      <xs:element name="body" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>
```

The note element is a **complex** type because it contains other elements. The other elements (to, from, heading, body) are **simple types** (string) because they do not contain other elements. Final XML code with reference to the XML Schema

```
 <?xml version="1.0"?>

<note
xmlns="http://www.cse.iitb.ac.in"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.cse.iitb.ac.in note.xsd">
  <to>John</to>
  <from>David</from>
  <heading>Reminder</heading>
  <body>Meeting is cancelled.</body>
</note>
```

3.3.3 RDF (Resource Description Framework)

Resource Description Framework (RDF)[RDF] is a graphical language used for representing information about resources on the web. It is a basic ontology language. RDF is written in XML. By using XML, RDF information can easily be exchanged between different types of computers using different types of operating systems and application languages. RDF was designed to provide a common way to describe information so it can be read and understood by computer applications. RDF descriptions are not designed to be displayed on the web. Data model for objects and relations between them, provides a simple semantics for datamodel. Data models can be represented in XML syntax. **Basic RDF code looks like**

Table 3.4: Student Information

Student Id	Name	Subject	Marks	Percentage
059600	John	Networks	40	80
059601	David	Networks	45	85

For the above data the RDF code is

```
 <?xml version="1.0"?>

<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:st="http://www.cse.iitb.ac.in/st#">

<rdf:Description
rdf:about="http://www.cse.iitb.ac.in/st/059600">
  <st:name>John</st:name>
  <st:subject>Networks</st:subject>
  <st:marks>40</st:marks>
  <st:percentage>80</st:percentage>
</rdf:Description>

<rdf:Description
rdf:about="http://www.recshop.fake/st/059601">
  <st:name>David</st:name>
  <st:subject>Networks</st:subject>
  <st:marks>45</st:marks>
  <st:percentage>90</st:percentage>
</rdf:Description>
</rdf:RDF>
```

The first line of the RDF document is the XML declaration. The XML declaration is followed by the root element of RDF documents: <rdf:RDF>.

The xmlns:rdf namespace, specifies that elements with the rdf prefix are from the namespace

`"http://www.w3.org/1999/02/22-rdf-syntax-ns#".`

The xmlns:cd namespace, specifies that elements with the cd prefix are from the namespace

`"http://www.iitb.ac.in/st#".`

The <rdf:Description>element contains the description of the resource identified by the rdf:about attribute.

The elements: <st:name>, <st:subject>, <st:marks>, etc. are properties of the resource.

RDF identifies with Uniform Resource Identifiers (URI) these are called resources. The base element of the RDF model is the triple: a subject linked through a predicate to object. We will say that <subject>has a property <predicate>valued by <object>

The RDF triple (S,P,O) can be viewed as a labeled edge in graph.

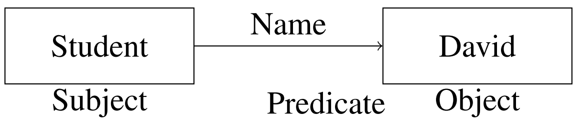


Figure 3.10: Predicate

RDFS (RDFSchema)

Vocabulary for describing properties and classes of RDF resources[Wikirdfs], with a semantics for generalization - hierarchies of such properties and classes. It defines a simple ontology that particular RDF documents may be checked against to determine consistency. Many RDFS components are included in the more expressive language Web Ontology Language (OWL).

3.3.4 OIL (Ontology Interchange Language)

OIL is also known as Ontology Inference Layer[OIL]. OIL is derived from RDFS. OIL is based on descriptive logic. Descriptive logic describes knowledge in terms of concepts and role restrictions that can automatically derive classification taxonomies.

OIL is better than its ancestors in the following ways. It has rich set of modeling primitives and nice ways to define concepts and attributes. The definitions of a formal semantics were included in OIL. Customized editors and interface engines for OIL also exist. Any RDFS ontology is a valid ontology in OIL and vice versa. Much of the work in OIL was subsequently incorporated into DAML+OIL and the Web Ontology Language (OWL).

3.3.5 OWL (Web Ontology Language)

In 2001, the W3C formed a working group called Web-Ontology (WebOnt) Working Group. The aim of this group was to make a new ontology markup language for the Semantic Web, called OWL (Web Ontology Language)[OWL]. OWL is used when the information contained in documents needs to be processed by application. OWL can be used to explicitly represent the meaning of terms in vocabularies and the relationships between the terms. It is a revised version of the DAML + OIL web ontology language. OWL adds more vocabulary for describing properties and classes.
Siblings of OWL are

1. OWL Lite

2. OWL DL

3. OWL Full

OWL Lite

OWL lite supports classification hierarchy and simple constraints. OWL Lite provides a quick migration path for thesauri and other taxonomies. OWL Lite has a lower formal complexity then OWL DL.

OWL DL

Maximum expressiveness while retaining computational completeness and decidable i.e. all computations will be finished in time. OWL DL is named due to its correspondence with Description Logic, and it includes all the OWL language constructs.

OWL Full

OWL Full gives syntactic freedom of RDF, with no computational guarantees. OWL Full allows an ontology to augment the meaning of the pre-defined (RDF or OWL) vocabulary

The following set of relations hold. Their inverses do not.

- Every legal OWL Lite ontology is a legal OWL DL ontology.
- Every legal OWL DL ontology is a legal OWL Full ontology.
- Every valid OWL Lite conclusion is a valid OWL DL conclusion.
- Every valid OWL DL conclusion is a valid OWL Full conclusion.

The choice between OWL-Lite and OWL-DL is based on whether the simple constructs of OWL-Lite is sufficient or not. Whereas the choice between OWL-DL and OWL-Full may be based upon whether it is important to be able to carry out automated reasoning on the ontology or whether it is important to be able to use highly expressive and powerful modeling facilities such as meta-classes (classes of classes).

OWL Full can be viewed as an extension to RDF. whereas OWL Lite and OWL DL can be viewed as an extension of a restricted view of RDF.

Every OWL (Lite, DL, Full) document is an RDF document and every RDF document is an OWL Full document. Only some RDF documents can be OWL lite or OWL DL.

Table 3.5: OWL Example.

Person	Animal	Relation
Rex	John	Pet

OWL code for the above table is:

```
[Namespaces:
  rdf        = http://www.w3.org/1999/02/22-rdf-syntax-n\#
  xsd        = http://www.w3.org/2001/XMLSchema#
  rdfs        = http://www.w3.org/2000/01/rdf-schema#
  owl        = http://www.w3.org/2002/07/owl#
  pp        = http://cohse.semanticweb.org/ontologies/people#
]

Ontology(
Class(pp: person)
Class(pp: animal)
ObjectProperty(pp:has_pet domain(pp:person) range(pp:animal))
                                  //Property
Individual(pp:Rex type(pp:dog) value(pp:is_pet_of pp:John))
                                  //Instance
)
```

3.4 Ontology Editors

Ontology editors are designed to assist in the creation or modification of ontologies. These editors are the applications which support one or more ontology languages. And some editors also have the facility to export from one to another ontology languages.

3.4.1 Ontolingua

The Ontolingua Server was the first ontology tool created[2]. It was developed in the Knowledge Systems Laboratory (KSL) at Stanford University. The public server is available at http://www-ksl-svc.stanford.edu:5915/, we must have an account to use the system. All work takes place with in a session, which has a duration and a description. Ontolingua uses Object-Oriented presentation, and full logical representation. The ontology is stored on the server. we can download our work to a local file system or we can email our work to any system. Ontolingua runs in port 5915.
Uses of Ontolingua are:

1. Runtime query
2. Translation

Runtime Query, remote applications may query an ontology server

- To determine if a term is defined
- To determine the relationship between terms
- To manipulate the contents of an ontology

Translation, from one language to another language. It includes the following challenges

- **Semantics -** ensure that the meaning is preserved.
- **Syntax -** ensure that target syntax is correct.
- **Style -** ensure that target idioms are preserved.

Ontolingua is not suitable for our system because

Ontolingua runs in port 5915. There is a firewall in our network which is blocking HTTP connections to ports other than port 80. So we were not able to open Ontolingua server.

[2]http://ksl.stanford.edu/software/ontolingua/

3.4.2 Protégé

Protégé is a free, open source ontology editor and a knowledge acquisition system[3]. It was developed by the Stanford Medical Informatics(SMI) at Stanford University. It is an opensource, standalone application with an extensible architecture. It is extensible in the sense that we can add plug-ins to the tool. It is written in Java and heavily uses Swings to create the complex user interface. Protégé ontologies can be exported into a variety of formats including RDF(S), OWL, and XML Schema. The Protégé platform supports two main ways of modeling ontologies

1. Protégé-OWL
2. Protégé-Frames

Protégé-OWL

The Protégé-OWL editor is an extension of Protégé that supports the Web Ontology Language (OWL). An OWL ontology may include descriptions of classes, properties and their instances. The Protégé-OWL editor enables users to:

1. Load and save OWL and RDF ontologies.
2. Edit and visualize classes, properties, and rules.
3. Define logical class characteristics as OWL expressions.

protégé-frames

The Protégé-Frames editor provides user interface to support users in constructing and storing domain ontologies. In this model, an ontology consists of a set of classes organized in hierarchical order to represent a domain's concepts. Classes are associated with a set of slots to describe their properties and relationships, and a set of instances of those classes.

Developing Ontology

To create Ontology using protégé user should have all the keywords and their relationships in hierarchical order. First of all install protégé in the system go to the classes tab. insert each keyword in hierarchical order. A sample computer networks ontology developed by our system using protégé is shown in the Figure B.1.

Reason why we have taken protégé:

- **Open-source:** Protégé is available as free software under the open-source *Mozilla Public License*.

[3] http://Protege.stanford.edu/

- **Extensible:** we can add the add-ons so that it will fit into our purpose.
- **Standalone:** Unlike other tools protégé can be downloaded to work with it. No internet is required for development of ontology.

3.4.3 WebODE

WebODE is a tool for building ontologies in the World Wide Web[4]. It was developed in the Artificial Intelligence Lab from the Technical University of Madrid(UPM). Web ODE can't be used as a standalone application, but we can use it as a Web server with a Web interface. Multiple concurrent users are allowed to work at the same time, proper synchronization and blocking mechanisms are provided. Ontology designers can make their best with this tool either working alone or in a team.

WebODE is based on a central ontology repository implemented using a relational database. A very simple and powerful mechanism to export and import ontologies using the XML standard is also supplied by default with the tool. Technical requirements

1. Browser (Internet Explorer 5.0 or above)

2. Java plug-in version 1.2.x must be installed in the browser

3. Configure the Web browser so that the pages are retrieved from the server every time they are visited. This can be done by navigating to "Tools / Internet Options / Internet Temporal Files / Configuration / Every Time the page is Visited".

4. Do not use the proxy for the WebODE URL. This option can be configured selecting: Tools / Internet Options/Connection / LAN Configuration.

Usage Login in this website http://www.oeg-upm.net/webode

New Ontology: Give a name and description for the ontology that is being created. Where name is compulsory and description is optional.

List of Ontologies: We can list the ontologies that we are able to access (and modify) by clicking on the list ontologies tab from the main menu.

Open Ontology: We can open ontologies, to insert new components, to update them and to simply remove some of them.

Export Ontology: We can export the ontology into several languages like UML, XML, RDF(S), OIL, DAML+OIL, OWL and so on, and we can get the target code.

[4]http://webode.dia.fi.upm.es/WebODEWeb/Documents/usermanual.pdf

Developing Ontology

First we have to register for WebODE, after registration we will get a username and password. Log on to the WebODE website http://www.oeg-upm.net/webode using the username and the password. Here also the keywords and the relationships between the keywords should be known prior to the ontology creation.

WebODE is not suitable for our system because

WebODE is not a standalone application. It is a web based application. And one more reason is WebODE doesn't support OWL2.

3.4.4 OntoStudio

OntoStudio is an modeling environment to create and maintain ontologies[5], with particular emphasis on rule-based modeling. It was originally developed for F-Logic but now also includes some support for OWL, RDF, and OXML. It also includes functions such as the OntoStudio Evaluator. Evaluator is used for the implementation of rules during modeling. It is the successor of OntoEdit.

Data Models

OntoStudio can be operated with a RAM-based- as well as database-data-model; therefore it is scalable and suitable for the modeling of large ontologies in business domain.

1. **RAM Data Model:**
   The complete ontology data is loaded into the main memory of the workstation. All changes made in the ontology are written in files when you save the new ontologies. It is appropriate for small and medium-size ontologies (dependent on main memory size)

2. **Database Data Model**
   Only the actual presented parts of the ontology are loaded from a database into main memory and will be written back to the database immediately when changes happen in the ontology. It is appropriate for large ontologies.

OntoStudio supports the ontology languages F-Logic, RDF/RDFS and OWL.

Developing Ontology

For creating ontologies using OntoStudio install OntoStudio in the system and run from the command prompt.

[5]http://www.ontoprise.de/en/home/products/ontostudio/

OntoStudio is not suitable for our system because

OntoStudio is not open-source software. It is licensed under ontoprise. It is a commercial modeling environment for the creation and maintenance of ontologies. So it is not helpful for our system. Summary of different ontology tools we studied, their comparison with respect to languages they support, mode of access, export and import options were given in the table - 3.6.

Table 3.6: Comparison of Ontology development tools, taken from [?]

	Ontolingua	**Protégé**	**WebODE**	**OntoStudio**
Developers	KSL (Stanford University)	SMI (Stanford University)	UPM	Ontoprise
Current Release and Tools	0.1.45(Aug 2003)	4.1Alpha(Mar 2010)	2.0(Dec 2002)	2.3.3(Dec 2009)
Pricing Policy	Free Web Access	Free Ware	Lincences	Freeware & Licences
Mode of Access	Web Access	Stand Alone	Web Access	Stand Alone
Export to Languages	CLIPS CML, ATP CML	XML, RDF(S), XMLSchema, OWL	XML, RDF(S), OIL, DAML+OIL	XML, RDF(S), F-Logic, OWL
Import from Languages	IDL KIF	XML, RDF(S), XMLSchema, OWL	XML, RDF(S)	XML, RDF(S), F-Logic, OWL
Ontology Library	Yes	Yes	No	Yes

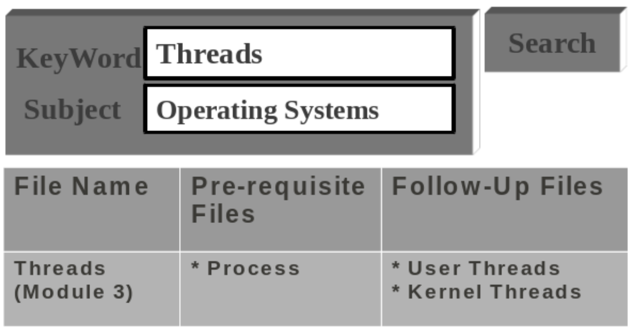
Chapter 4

System Overview

4.1 Problem Statement

Given a repository of lecture notes for a particular subject (or the soft-copy of a text book), our aim is to build a system that provides the user with: (i) a mechanism to query for a desired topic, (ii) identify the pre-requisites and follow-ups for the topic and (iii) a dependency graph of the pre-requisites and follow-ups of all topics in the subject.

Suppose user wants to learn about particular topic say Threads, of particular subject say Operating system, we will provide user with the reading material (*The PDF file which contains the topic Threads*) and we will recommend the user with prior knowledge(*Link to the previous PDF files*) required by the user to start his desired topic and also the advanced topics(*Link to the next PDF files*) which he can cover after finishing the topic. The sample user interface for the desired solution is given in the Figure 4.1.



KeyWord: Threads

Subject: Operating Systems

File Name	Pre-requisite Files	Follow-Up Files
Threads (Module 3)	* Process	* User Threads * Kernel Threads

Figure 4.1: Desired solution

The desired system will also provide a dependency graph for the user so that the user can have a conceptual view of the course. A sample dependency graph generated manually and drawn using DOT language[1] is shown in Figure 4.2.

[1]http://en.wikipedia.org/wiki/DOT_language

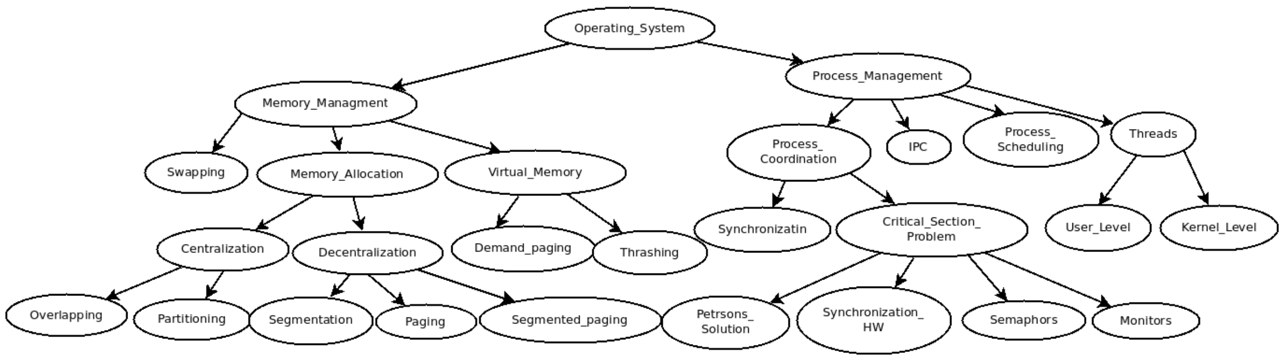


Figure 4.2: Dependency Graph for operating system

4.2 Proposed Solution

As the examined solutions mentioned in the previous chapter, which are available for generation of ontology, are not useful directly or indirectly for our purpose we propose the following solution.
Use some Natural Language processing tool to extract the keywords from the text files. Then find out the parts of speech, Noun-verb-Noun phrases. Find out the concepts and classes their dependencies and relationships. This can be done by many methods like finding the word frequencies, Noun verb patterns etc. Then build domain concepts and relationships and construct the dependency graph in hierarchical order which is the final formal domain ontology. The complete process is shown in the Figure 4.3, Taken from[Grub95]. Now take the User query and provide the output with help of the ontology.

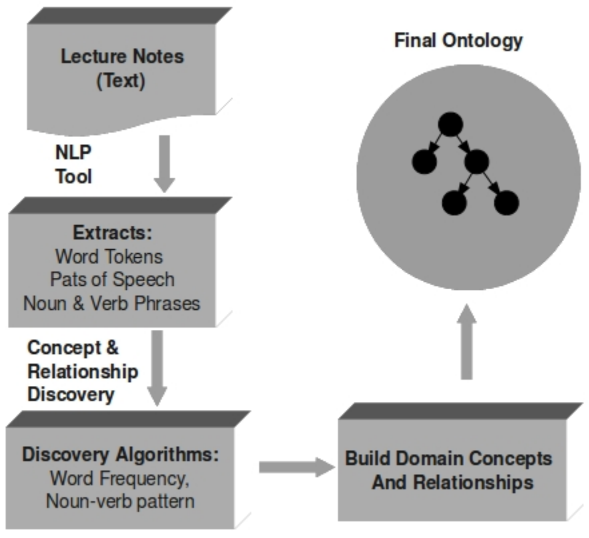


Figure 4.3: Ontology Development from Text, taken from[Grub95]

4.3 Solution Outline

System-1

The steps for our solution are as follows and also shown in Figure 4.4:

1. Convert the given PDF files into text, and index the text files.
2. Extract the keywords from the text files.
3. *For each keyword*: Identify its relation with other keywords. If there is a relation, then determine whether it is a pre-requisite or a follow-up.
4. Construct and store the ontology with the identified relations, in the form of a dependency graph in a hierarchical order.
5. Given a user query, lookup the ontology and provide the pre-requisites (ancestors) and follow-ups (descendants), as a reply to the query.

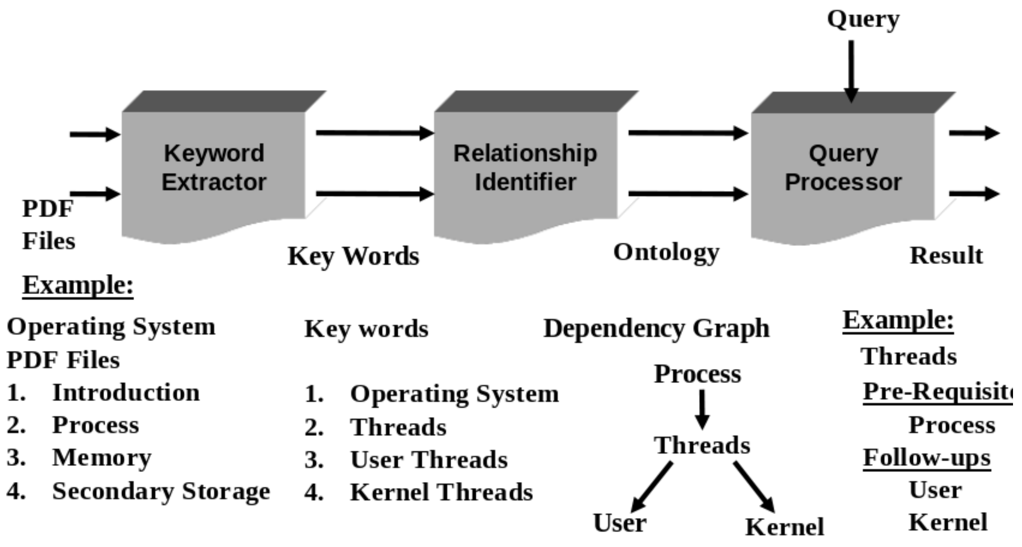


Figure 4.4: System overview of system-1

System-2

We modified our basic system in order to increase the performance of legitimate keyword extraction accuracy and system performance. Modified algorithm is shown in the Figure - 4.5. The modifications that were done to the above system are.

- After converting the PDF files to text files, we identified the Name entities and we removed the name entities from the corpus.
- To increase the number of legitimate keywords we used stemming (Root word identification) technique.
- Now extract the keywords from the stemmed corpus
- We modified the above proposed Apriori algorithm in order to increase the efficiency of relationship identification.

Other steps are same as our first system such as construction and storage of final ontology and query module.

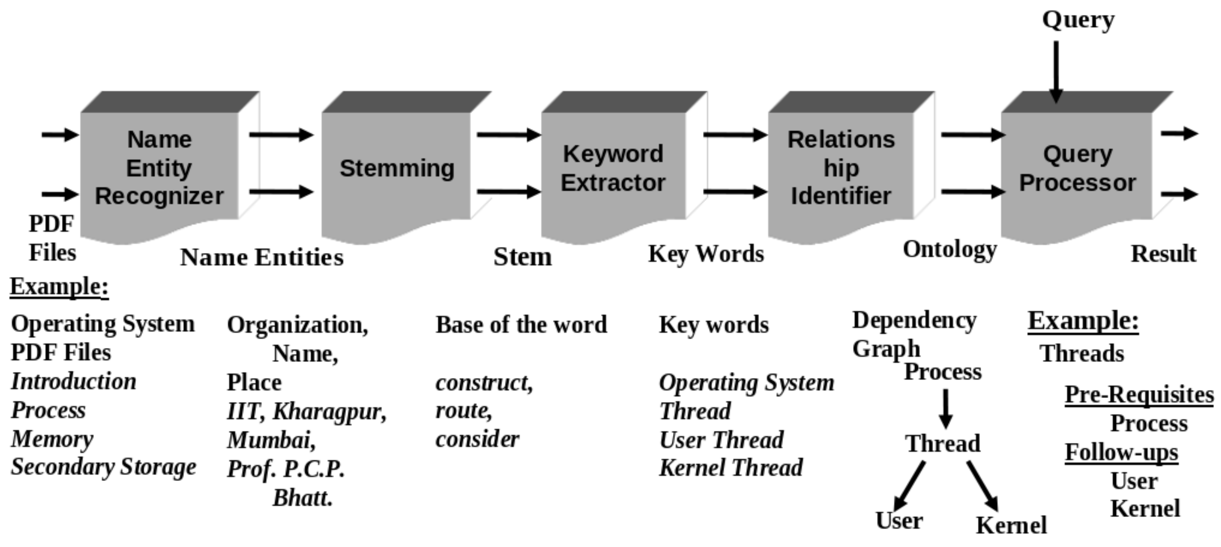


Figure 4.5: System overview of system-2

Chapter 5

Implementation Details

5.1 system-1

The implementation of our system has the following five phases:

1. Parsing: Parse the PDF files of the course to get the corpus text files, using PDFBox [PDFB].

2. Indexing: Index the Text files, using Lucene[WikiLuc].

3. Keyword Extraction: Extract the keywords, using tf-idf weighting scheme[RAM03].

4. Ontology Construction: Find the relations between the keywords, using the apriori algorithm [APR94].

5. Dependency Graph Generation: Generate the dependency graph for the whole course, using DOT [DOT], and ontology using OWL language[OWL].

The flow of input to output of system-1 is shown in Figure 5.1 and the details are described in the subsequent sections.

5.1.1 Parsing

We made use of a Nutch utility called PDFbox [PDFB] to parse the PDF files and to convert them into text (as required by our indexing utility, Lucene). In case there is only one big PDF file containing all the topics (such as soft-copy of a book), we used PDFBox also to divide it into multiple PDF files, since our algorithm requires multiple PDF files for identifying relations.

5.1.2 Indexing

Lucene[WikiLuc] is embedded in our system to index and search the given text documents. It lets one add indexing and searching capabilities to the application. Lucene can index any

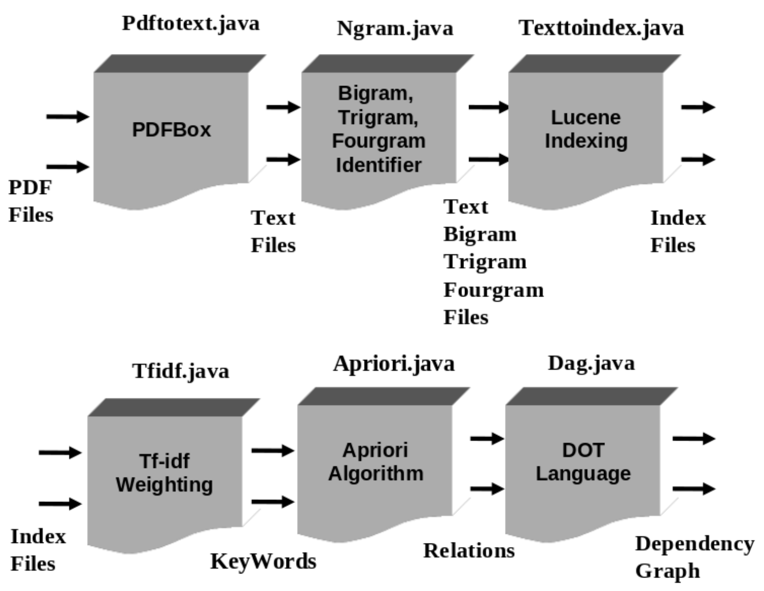


Figure 5.1: System Design

data that can be converted to textual format, and make it searchable. We used Lucene index to search and to calculate the tf-idf weight of each term (as described below), and also to identify the relationship between two keywords, i.e., whether one is a pre-requisite for other or a follow-up to the other.

5.1.3 Keyword Extraction

Keyword extraction is the process of extracting important phrases which can summarize the meaning of a document. Keywords can be extracted using linguistic techniques or machine learning techniques. Linguistic techniques make use of part-of-speech tagging or phrase chunking [Wikikey]. On the other hand, machine learning techniques use statistical or probabilistic data for keyword extraction. Machine learning based keyword extraction is once again divided into supervised and unsupervised techniques. Supervised techniques require some data for which the keywords are known (this is called training data) for its operation, while unsupervised techniques do not require any training data.

We used tf-idf (an unsupervised technique), to identify terms with high relevance to the document. We used “topia”[Topia] (which makes use of part-of-speech (POS) tagging technique), to find other keywords which are missed by tf-idf algorithm, if any.

Term Frequency Inverse Document Frequency (tf-idf)[RAM03]

Term frequency inverse document frequency (tf-idf) is defined as the number of occurrences of a term in a given document multiplied by the inverse of the number of documents where the term appears. Given a document collection D, a word w, and an individual document $d \in D$

$$w_d = f_{w,d} * log(|D|/f_{w,D})$$

where

- $f_{w,d}$ equals the number of times w appears in d,
- $|D|$ is the size of the corpus (number of documents), and
- $f_{w,D}$ equals the number of documents in which w appears in D.

Logarithm of document frequency in the above formula is used for smoothing purpose. The tf-idf value is

- high when a term occurs many times within a small number of documents,
- low when the term occurs fewer times in a document, or occurs in many documents,
- lower when the term occurs in virtually all documents.

Using tf-idf weighting scheme, we scored each word (unigram, bigram, trigram, and fourgram) in the given text corpus. Then we found out the top unigrams, bigrams, trigrams, and fourgrams according to the tf-idf weights. *Ngrams* are groups of n written letters, n syllables, or n words. In this way, we extracted top keywords for the given text.

We defined our own set of stop words in order to increase the accuracy of the extracted keywords. Stop words are common words like numbers, digits, articulations, conjunctions etc. We did not allow any stop words in bigrams. We allowed stop words as a second word for conjunction purpose in trigrams. In fourgrams, the third and the fourth words can be stop words but the first and the last words cannot be stop words. All the keywords extracted are stored in a file called `tfidf.txt`.

We performed experiments to determine the optimal number of unigrams, bigrams, trigrams, fourgrams, and total number of keywords to include. These are discussed in the evaluation section.

5.1.4 Ontology Construction

To identify the relation between different keywords and to construct the ontology, we used apriori algorithm, a classical algorithm for learning association rules.

Apriori Algorithm

"Given a set of documents, generate all association rules that have support and confidence greater than the user-specified minimum support and minimum confidence respectively"[APR94]. Where *Association Rule:* $i_1, i_2, ..., i_k \rightarrow j$ means, "if a document contains all of $i_1, ..., i_k$

then it is likely to contain j". *Support* is the number of documents containing all the words $i_1, i_2, ..., i_k, j$ and *Confidence* of this association rule is the probability of j given $i_1, ..., i_k$.

We modified the apriori algorithm according to our requirement. Our modified version is as follows:

- **Step 1:** Read documents and count the occurrences of each item. It requires only memory proportional to the number of words in the documents. This step was modified by using tf-idf weight.

- **Step 2:** Read documents again and count only those pairs which were found in Step 1 to be frequent. So if we find n frequent keywords, then we will have $n(n-1)/2$ pairs. Now, find out the *frequent wordsets* with the given confidence. The *frequent wordsets* establish an *Association* between the two words.

The *support* is a configurable parameter. If the support is less (minimum number of documents in which the pair of words should be repeated), then the algorithm will find more number of relations. On the other hand, if the support is more then, the algorithm will identify less number of relations. We considered the confidence as 0%. That is, if the support is satisfied, then we consider it as a relation, though other documents may contain only word-1 but not word-2.

For each pair of keywords in `tfidf.txt`, we calculated the frequency among different documents, and listed the pair of words having greater frequency than the support provided as relations. All the relations are stored in a file called `relation.dot` in a form which can be read by DOT language.

5.1.5 Generating the Dependency Graph & ontology

Having found the relations between the keywords, we show them in a graphical representation, i.e., the dependency graph. The purpose of creating the dependency graph is to let the users decide what they should know before learning any specific topic. Using DOT language, we constructed a directed acyclic graph for the relations that we identified above, and converted the file `relation.dot` into `dag.pdf` which is the dependency graph. From the above identified relations we construct ontology using OWL language, which can then be exported to protégé, so that we can add or delete concepts in future.

5.2 System - 2

The flow of input to output of system-2 is shown in Figure 5.2 and the details are described in the subsequent sections.

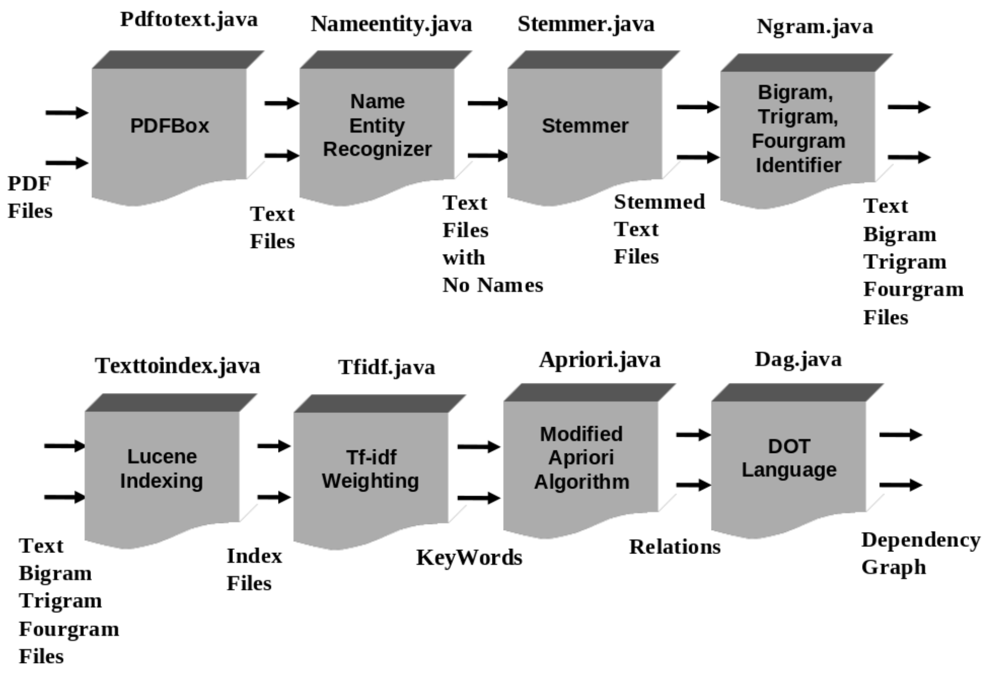


Figure 5.2: System Design

5.2.1 Stemming

We used stemming to decrease the number of words to be indexed and also to increase the performance of keyword identification. If the root word is present in different places in different forms then it is difficult to identify it as a keyword using Tf-idf algorithm. So we used stemming to identify the stem of the words and replaced different form of the words by its stem. This process reduces the number of words to be indexed and helps identifying the legitimate keywords.

We converted the given text files into vectors of *words* or *terms*. *Words* or *terms* are the root words derived from the original word after removal of the suffix by the stemming algorithm. Then we identified the keywords using keyword extraction algorithm.

To improve the number of legitimate keyword identification and to increase the performance of the system, we used a stemming. "Stemming is the process for reducing inflected (or sometimes derived) words to their stem, base or root form"[Wikistem]. Terms having the same root (stem) will have similar meanings. Consider the example of root word *create*:

- *create, created, creating, creates, creative, creativity, creatively, creation, creationism, creationisms, creationist, creationists, creations, creativeness, creator, creators....*

If we stem all these words then we will get only one root word that is *create* which will decrease the number of unique words in the given corpus, resulting efficient keyword extraction.

As morphological variant words are grouped together into a single term, it will reduce the number of words in the system and hence increase the performance of the system. There are many methods for stemming some uses linguistic dictionary for stemming whereas others use algorithmic rules based on suffixes list to get the root word. Two major assumptions that were made to use stemming are[Stem80]:

Suffixes are removed only to increase the performance of the system (accuracy of the keywords), and not as a linguistic exercise. That is suffixes are removed automatically (Mechanically) without using any linguistic means. Example: The algorithm will stem *operating* to *operate*, which is not acceptable in the domain of operating system.

100% accuracy cannot be achieved by using suffix list approach. For example *Probe* and *probation* have different meanings and stemming will not be a good option.

The above mentioned problems were some what subsidized in our algorithm by defining a fixed set of words for which stemming is not required.

For stemming we used a modified version of the algorithm written by M.F.Porter[Stem80]. In this algorithm linguistic dictionary is not used instead suffix list is used. The main motto of using this algorithm is to increase the performance of the system not linguistic accuracy. The modified version of the stemming algorithm is described in the following subsection.

Stemming Algorithm

Step-1 This step deals with plurals (-s,-ies, -sses) and past participles (-ed,-eed,-ing).

Step-1a Stem the plurals to its base form. (-s, -ies, -sses)

Rule			*Example*		
SSES	$\rightarrow$	*SS*	*accesses*	$\rightarrow$	*access*
IES	$\rightarrow$	*Y*	*carries*	$\rightarrow$	*carry*
SS	$\rightarrow$	*SS*	*access*	$\rightarrow$	*access*
S	$\rightarrow$	ϵ	*works*	$\rightarrow$	*work*

Step-1b Stem the past participles to its base form. (-ed, -eed, -ing)

Rule			*Example*			*Remarks*
EED	$\rightarrow$	*EE*	*freed*	$\rightarrow$	*free*	*If there are more than one VC pair*
			seed	$\rightarrow$	*seed*	*If there is only one VC pair*
ED	$\rightarrow$	ϵ	*worked*	$\rightarrow$	*work*	*If the stem contains at least one vowel* $(*v*)$*ED*
			red	$\rightarrow$	*red*	*If there is no vowel in the stem*
ING	$\rightarrow$	ϵ	*working*	$\rightarrow$	*work*	*If the stem contains at least one vowel* $(*v*)$*ING*
			string	$\rightarrow$	*string*	*If there is no vowel in the stem*

Step-2 After checking for plurals and past participles, map double suffices to single ones. This step is checked only when there is at least one VC pair, where V is combination of vowels and C is a combination of consonants.

Rule			*Example*		
ATIONAL	→	*ATE*	*relational*	→	*relate*
TIONAL	→	*TION*	*conditional*	→	*condition*
			rational	→	*rational*
ENCI	→	*ENCE*	*valenci*	→	*valence*
ANCI	→	*ANCE*	*hesitanci*	→	*hesitance*
IZER	→	*IZE*	*digitizer*	→	*digitize*
ABLI	→	*ABLE*	*conformabli*	→	*conformable*
ALLI	→	*AL*	*radicalli*	→	*radical*
ENTLI	→	*ENT*	*differentli*	→	*different*
ELI	→	*E*	*vileli*	→	*vile*
OUSLI	→	*OUS*	*analogousli*	→	*analogous*
IZATION	→	*IZE*	*vietnamization*	→	*vietnamize*
ATION	→	*ATE*	*predication*	→	*predicate*
ATOR	→	*ATE*	*operator*	→	*operate*
ALISM	→	*AL*	*feudalism*	→	*feudal*
IVENESS	→	*IVE*	*decisiveness*	→	*decisive*
FULNESS	→	*FUL*	*hopefulness*	→	*hopeful*
OUSNESS	→	*OUS*	*callousness*	→	*callous*
ALITI	→	*AL*	*formaliti*	→	*formal*
IVITI	→	*IVE*	*sensitiviti*	→	*sensitive*
BILITI	→	*BLE*	*sensibiliti*	→	*sensible*

Step-3 this step deals with -ic-, -full, -ness etc, Similar to Step-2 this step is also considered only when there is at least one VC pair, where V is combination of vowels and C is a combination of consonants.

Rule			*Example*		
ICATE	→	*IC*	*triplicate*	→	*triplic*
ATIVE	→	ϵ	*formative*	→	*form*
ALIZE	→	*AL*	*formalize*	→	*formal*
ICITI	→	*IC*	*electriciti*	→	*electric*
ICAL	→	*IC*	*electrical*	→	*electric*
FUL	→	ϵ	*hopeful*	→	*hope*
NESS	→	ϵ	*goodness*	→	*good*

Step-4 Last step is to takes off -ant, -ence etc., Unlike Step-2 and Step-3 this Step is checked against more than one VC pair, where V is combination of vowels and C is a combination of consonants.

Rule			*Example*		
AL	$\rightarrow$	ϵ	*revival*	$\rightarrow$	*reviv*
ANCE	$\rightarrow$	ϵ	*allowance*	$\rightarrow$	*allow*
ENCE	$\rightarrow$	ϵ	*inference*	$\rightarrow$	*infer*
ER	$\rightarrow$	ϵ	*airliner*	$\rightarrow$	*airlin*
IC	$\rightarrow$	ϵ	*gyroscopic*	$\rightarrow$	*gyroscop*
ABLE	$\rightarrow$	ϵ	*adjustable*	$\rightarrow$	*adjust*
IBLE	$\rightarrow$	ϵ	*defensible*	$\rightarrow$	*defens*
ANT	$\rightarrow$	ϵ	*irritant*	$\rightarrow$	*irrit*
EMENT	$\rightarrow$	ϵ	*replacement*	$\rightarrow$	*replac*
MENT	$\rightarrow$	ϵ	*adjustment*	$\rightarrow$	*adjust*
ENT	$\rightarrow$	ϵ	*dependent*	$\rightarrow$	*depend*
*(*S or *T)) ION*	$\rightarrow$	ϵ	*adoption*	$\rightarrow$	*adopt*
OU	$\rightarrow$	ϵ	*homologou*	$\rightarrow$	*homolog*
ISM	$\rightarrow$	ϵ	*communism*	$\rightarrow$	*commun*
ATE	$\rightarrow$	ϵ	*activate*	$\rightarrow$	*activ*
ITI	$\rightarrow$	ϵ	*angulariti*	$\rightarrow$	*angular*
OUS	$\rightarrow$	ϵ	*homologous*	$\rightarrow$	*homolog*
IVE	$\rightarrow$	ϵ	*effective*	$\rightarrow$	*effect*
IZE	$\rightarrow$	ϵ	*bowdlerize*	$\rightarrow$	*bowdler*

Complex suffixes are removed step by step. For example, *computerizations* is stripped to *computerization* in Step-1, then to *computerize* in Step-2, and to *computer* in Step-3

5.2.2 Name Entity Recognizer

Driven idea behind using named entity recognition is, named entities do not have any pre-requisites and follow-ups. A person, place or an organization will not have any pre-requisites and follow-ups.

Named Entity Recognition (NER) involves identifying named entities (proper nouns) and classifying them as person-name, location, organization, time, date, etc. The categories into which proper nouns are to be classified vary according to the applications, but common categories include, person names, location and organization names, medicine names, disease names and so on. In our case we considered only three types of named entities they are person names, location (Place) and organization. We identified these three categories of named entities and removed them from the corpus.

Consider the example:

```
My name is Neelamadhav.  I go to IITBombay, which is in Mumbai.
I study computer science.  I grew up in Parvathipuram.
```

This can be tagged as:

```
My name is <PERSON>Neelamadhav</PERSON>. I go to
<ORGANIZATION>IITBombay</ORGANIZATION>, which is in
<LOCATION>Mumbai</LOCATION>.  I study computer science.
I grew up in <LOCATION>Parvathipuram</LOCATION>.
```

Applications of NER

1. NER is very useful for search engines. NER helps in structuring textual information, and structured information helps in efficient indexing and retrieval of documents for search.

2. Before reading an article, if the reader could be shown the named entities, the user would be able to get a fair idea about the contents of the article.

3. Automatic indexing of Books: Most of the words indexed in the back index of a book are Named Entities.

4. Useful in Biomedical domain to identify Proteins, medicines, diseases, etc.

Named entities can be identified using supervised approaches or rule based approaches. Supervised approaches uses tagged data for which the named entities are already tagged, this data is known as training data. On the other hand rule based approaches use some predefined rules to identify the named entities. We used a CRF-based Named Entity Recognizer (NER) system[NER05] (Supervised Approach), developed by Stanford University to recognize the named entities and we removed them from the corpora.

CRF-based NER

We used Stanford NER mainly because of the following reasons

1. The NER is trained on CoNLL, MUC and ACE English training data

2. By default it recognizes the entities: Person, Location, Organization, Which we need for our experiment

3. Finally the NER is trained on both British and American newswire, so robust across both domains

5.2.3 Ontology Construction

To identify the relation between different keywords and to construct the ontology, we modified the previous defined apriori algorithm. Here we count number of lines which contain the pair of key words instead of number of documents. There is a possibility this will give more legitimate relations. . Because nearby terms have a strong relationship.

Modified Apriori Algorithm

Association Rule: $i_1, i_2, ..., i_k \rightarrow j$ is formed only, “if a line instead of document contains all of $i_1, ..., i_k$ then it is likely to contain j”. *Support* is the number of lines containing all the words $i_1, i_2, ..., i_k, j$ and *Confidence* of this association rule is the probability of j given $i_1, ..., i_k$.

Chapter 6

Evaluation

We used Computer Networks, Operating Systems, System analysis and Design, Software Engineering, Embedded Systems, Numerical Analysis, and Cryptography courses of NPTEL courseware repository for our testing purpose. The dependency graph for Computer Networks course is shown in the Figure 6.5. The graphs were generated using DOT language. DOT language orders the nodes in such a way that, node with less number of incoming edges comes at the top levels, and nodes with more number of incoming edges in the next levels. So, the order shown in the graph is not the exact output of our system, it is a graphical simulation of the output. In the dependency graph shown in Figure 6.5, concepts like *tcp, ospf, rip* are the keywords identified by our system. For a particular node the ancestors are pre-requisites and descendants are follow-ups. For example, the pre-requisites of the concept *tcp* is *topology* and the follow-up topic is *congestion control*.

We evaluated our system by manually comparing its results with the results given by an expert. We compared the keywords generated by our program with the expert generated keywords. Similarly we compared the relations generated by our system with those of the expert generated relations. We used `precision`, `recall` and `confusion matrix` to find the accuracy of the system. Formula for Precision and Recall is given in the following equations. Here, R_w, P_w denotes `recall`, and `precision`, respectively, for the keywords identified. Whereas, R_r, P_r denote `recall`, and `precision`, respectively, for the relations determined. `Precision` is, the correct keywords given by system with respect to the expert results. On the other hand, `recall` is, the correct keywords given by system with respect to keywords given by the system.

6.1 Precision and Recall

Let

- W_e and R_e denote the total number of keywords and relations suggested by the expert, respectively,

- W_s and R_s denote the total number of keywords and relations generated by our system, respectively, and
- W_c and R_c denote the common keywords and relations in both expert given and system generated, respectively, then

Recall (R) is the ratio of the number of relevant keywords/relations retrieved to the total number of keywords/relations suggested by the expert. It is usually expressed as a percentage.

$$R_w = \frac{W_c}{W_e} * 100\% \qquad R_r = \frac{R_c}{R_e} * 100\%$$

Precision (P) is the ratio of the number of relevant keywords/relations retrieved to the total number of keywords/relations identified by the system. It is usually expressed as a percentage.

$$P_{pw} = \frac{W_c}{W_s} * 100\% \qquad P_r = \frac{R_c}{R_s} * 100\%$$

`Confusion matrix` in our case contains keywords/relations suggested by experts and keywords/relations identified by our system. It contains two rows and two columns, which describes `true positive, true negative, false positive` and `false negative`. `Confusion matrix` relevant to our system is shown in the following table 6.1

		System Results	
		Positive	Negative
Expert Results	True	True Positive	False Negative
	False	False Positive	True Negative

Table 6.1: Confusion Matrix

Where

True Positive is the number of correct keywords/relations that were correctly identified,

False Positive is the number of incorrect keywords/relations that were incorrectly classified as positive,

True Negative is the number incorrect keywords/relations that were identified as negative, it is difficult to identify Negative keywords as we don't have the list of negative keywords.

False Negative is the number of correct keywords/relations that were incorrectly classified as negative.

For example, for a given subject as shown in the following Figure 6.2, let w, x, y, and z be the keywords identified by an expert, and let x, y, and p be the keywords identified by our system. In this case, the confusion matrix is as follows. True Positive is $\{x, y\}$, so the `recall` is $|\{x, y\}|/|\{w, x, y, z\}| = 2/4 = 0.5$, whereas the `precision` is $|\{x, y\}|/|\{x, y, p\}| = 2/3 = 0.66$.

		System Results(x, y, p)	
		Positive	Negative
Expert Results (w, x, y, z)	True	x, y	w, z
	False	p	-

Figure 6.1: Confusion matrix for the example

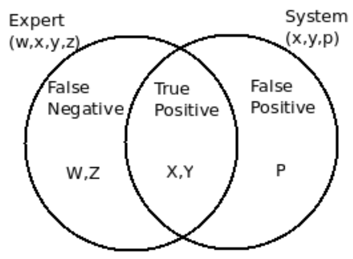


Figure 6.2: Classification Diagram

6.2 Performance Analysis

We manually identified the keywords for all the subjects, and then compared them with those of the results generated by our system. The results are given in the following tables. Table 6.2 shows the results for Computer Networks and Figure 6.3 is the graph for Computer Networks with respect to the results. In the same way, Table 6.3 and Figure 6.4 show the results for Operating Systems.

As `recall` increases `precision` decreases, conversely if `precision` increases `recall` decreases.

From the graph, we can observe that with increase in the number of system generated keywords, `recall` (R) increases. This is because, the expert given keywords (W_e) are fixed. At the same time with increase in number of keywords, `precision` (S_p) also increases up to a certain value. After this value if we further increase the number of keywords, it will result in more number of spurious keywords, than the number of legitimate keywords. So, `precision` will start decreasing with increase in number of system generated keywords after this maximum value. We considered the value where `precision` (P) is maximum as the optimum number of keywords. In the domain of "Computer Networks" the optimum value for number of keywords is 140. At this point 63.84% ($\frac{R_c}{R_s}$) of system answers are correct which happens to be finding 48.26% ($\frac{R_c}{R_e}$) of all correct answers. Similarly for "Operating System" domain the optimum value of keywords is 130.

6.3 Results of System -1

Table 6.2: Results for Computer Networks

keywords	Recall(%)		Precision(%)	
	R_w	R_r	P_w	P_r
98	19.76	14.89	34.69	29.93
136	47.09	42.47	59.56	54.79
155	48.84	43.84	54.19	51.32
175	51.16	46.57	50.29	47.21
195	52.33	47.29	46.15	41.82
234	57.56	50.18	42.31	38.19
243	57.56	50.18	40.74	36.88
252	57.56	50.18	39.29	35.43
262	57.56	50.18	37.79	33.16
272	57.56	50.18	36.40	32.74

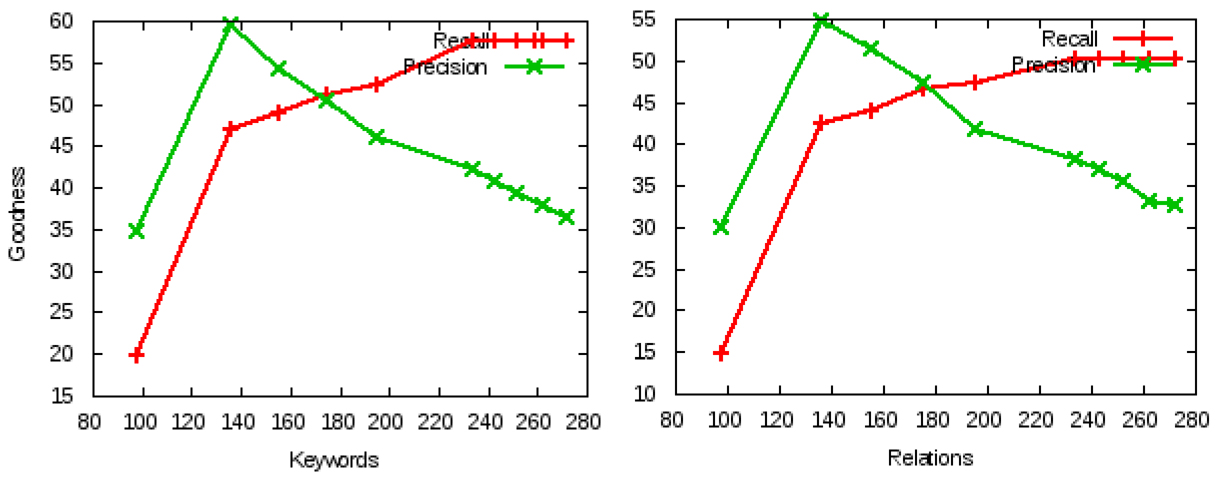


Figure 6.3: Computer Networks

Table 6.3: Results for Operating Systems

keywords	Recall(%)		Precision(%)	
	R_w	R_r	P_w	P_r
92	27.10	23.48	31.52	27.34
125	57.01	53.18	48.80	44.03
144	60.75	56.97	45.14	42.73
163	67.29	63.09	44.17	41.26
181	67.29	63.09	39.78	37.14
218	70.09	67.87	34.40	31.17
228	71.03	68.28	33.33	30.06
237	71.03	68.28	32.07	29.93
245	71.03	68.28	31.02	28.76
255	71.03	68.28	29.80	26.88

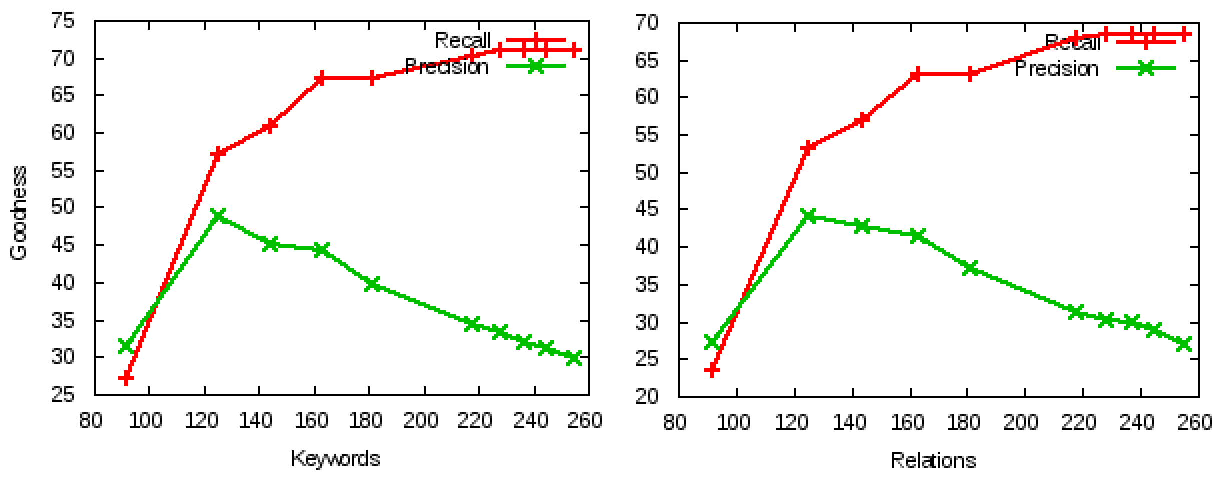


Figure 6.4: Operating Systems

Table 6.4: Results for Software Engineering

keywords	Recall(%)		Precision(%)	
	R_w	R_r	P_w	P_r
94	25.8	21.12	34.04	30.27
131	64.52	57.29	61.07	54.09
149	68.55	59.81	57.05	51.34
167	74.19	64.33	55.09	50.01
186	76.61	67.18	51.08	48.76
223	79.84	70.94	44.39	40.32
230	81.45	73.15	43.91	39.16
240	81.45	73.15	42.08	38.25
249	81.45	73.15	40.56	36.64
259	81.45	73.15	39	36.11

Table 6.5: Results for Cryptography

keywords	Recall(%)		Precision(%)	
	R_w	R_r	P_w	P_r
100	31.85	27.19	43.00	39.26
139	58.52	55.24	56.83	53.74
155	62.22	58.93	54.19	51.09
173	70.37	66.38	54.91	51.29
193	70.37	66.38	49.22	44.96
232	71.11	67.94	41.38	37.86
242	71.11	67.94	39.67	34.62
251	71.11	67.94	38.25	34.13
261	71.85	68.06	37.16	33.47
271	71.85	68.06	35.79	32.15

Table 6.6: Results for Embedded Systems

keywords	Recall(%)		Precision(%)	
	R_w	R_r	P_w	P_r
97	28.67	24.47	42.27	39.07
136	49.65	46.09	52.21	48.68
153	55.24	51.23	51.63	48.09
172	55.94	51.84	46.51	43.16
192	55.94	51.84	41.67	37.56
230	58.74	54.95	36.52	32.62
239	60.84	56.36	36.40	32.24
249	60.84	56.36	34.94	31.08
258	60.84	56.36	33.72	29.51
268	60.84	56.36	32.46	28.46

Table 6.7: Results for Numerical

keywords	Recall(%)		Precision(%)	
	R_w	R_r	P_w	P_r
94	38.70	34.69	24.49	21.38
131	67.74	64.08	30.66	26.74
149	72.58	68.86	29.03	25.86
167	79.03	76.28	28.16	25.04
186	79.03	76.28	25.39	22.91
223	83.87	79.81	22.71	19.27
230	85.48	82.05	22.18	19.01
240	87.10	84.46	21.69	18.67
249	87.10	84.46	21.01	18.03
259	87.10	84.46	20.22	17.75

Table 6.8: Results for System Analysis And Design

keywords	Recall(%)		Precision(%)	
	R_w	R_r	P_w	P_r
99	23.52	19.46	28.28	24.56
138	52.10	48.34	44.93	40.83
158	56.30	53.78	42.41	39.06
177	58.82	54.91	39.55	35.89
197	58.82	54.91	35.53	31.79
236	59.66	55.24	30.08	27.11
246	63.03	59.86	30.49	27.34
255	63.03	59.86	29.41	26.84
265	63.03	59.86	28.30	26.03
275	63.03	59.86	27.27	25.09

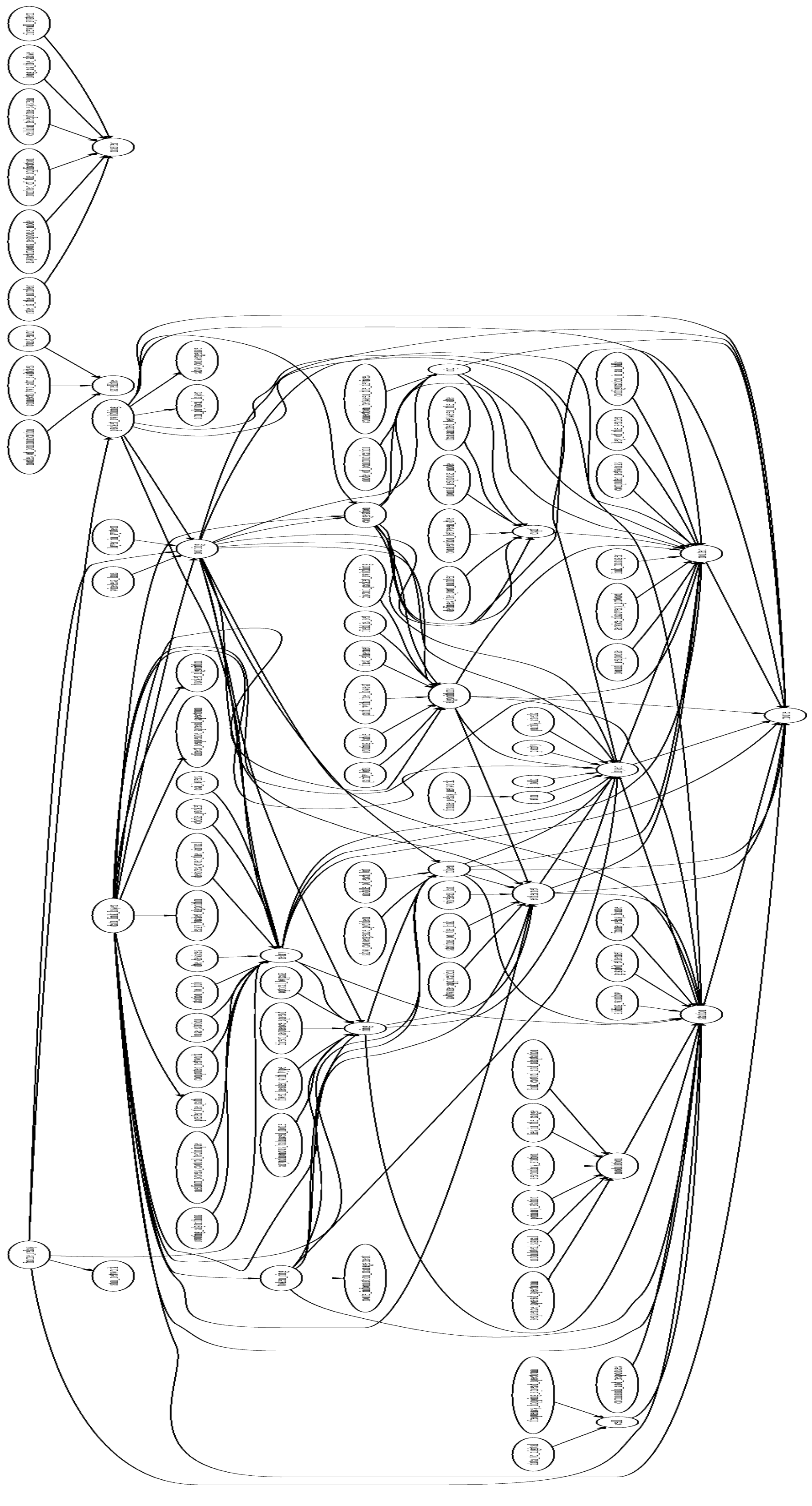


Figure 6.5: DAG for Computer Networks by System-1

6.4 Results of System - 2

Table 6.9: Results for Computer Networks

keywords	Recall(%)		Precision(%)	
	R_w	R_r	P_w	P_r
97	45.93	42.19	81.44	78.86
135	52.91	48.26	67.41	63.84
153	54.65	51.09	61.44	57.32
173	56.98	53.46	56.65	53.21
193	60.47	56.94	53.89	50.82
232	65.12	61.77	48.28	45.19
241	65.70	61.98	46.89	43.88
251	68.02	64.28	46.61	43.43
260	70.93	67.35	46.92	43.56
270	72.67	69.33	46.30	43.16

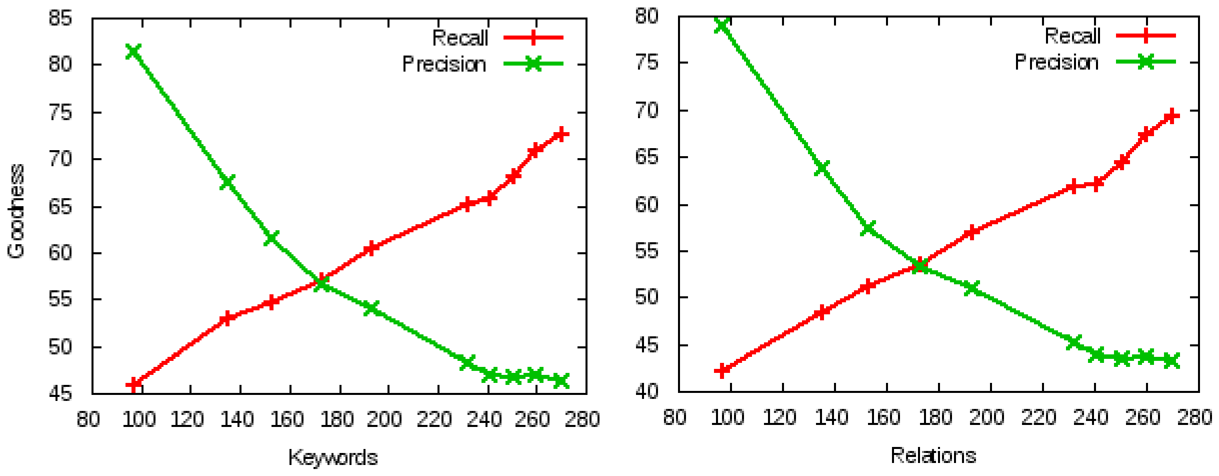


Figure 6.6: Computer Networks

Table 6.10: Results for Operating Systems

keywords	Recall(%)		Precision(%)	
	R_w	R_r	P_w	P_r
98	42.06	34.10	53.06	48.52
136	54.21	50.01	47.79	44.80
154	57.94	52.75	44.81	44.14
173	68.22	65.29	46.24	45.17
193	69.16	66.29	41.97	42.78
228	73.83	70.09	37.72	38.40
238	73.83	70.09	36.13	37.33
248	75.70	71.03	35.48	37.07
258	78.50	75.03	34.88	37.02
266	80.37	76.58	34.96	36.80

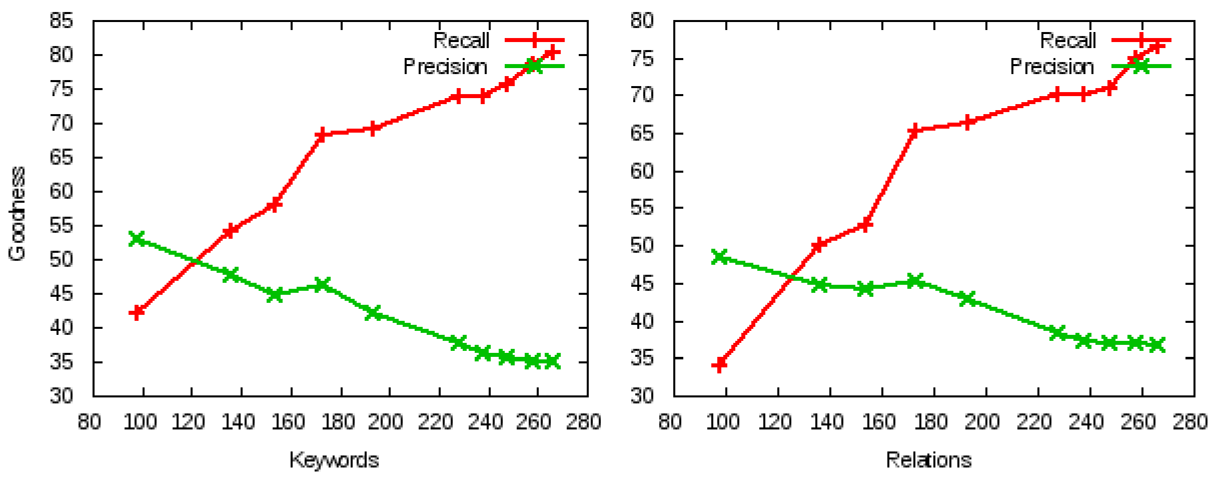


Figure 6.7: Operating Systems

Table 6.11: Results for Software Engineering

keywords	Recall(%)		Precision(%)	
	R_w	R_r	P_w	P_r
95	59.68	55.37	77.89	73.22
131	70.97	67.46	67.18	64.94
149	76.61	73.19	63.76	60.28
167	82.26	79.16	61.08	57.09
186	82.26	79.16	54.84	51.14
226	86.29	83.48	47.35	44.24
232	87.10	84.27	46.55	43.67
241	90.32	86.74	46.47	43.32
249	91.94	86.98	45.78	42.18
257	92.74	88.45	44.75	41.29

Table 6.12: Results for Cryptography

keywords	Recall(%)		Precision(%)	
	R_w	R_r	P_w	P_r
100	54.81	51.19	74.00	70.26
97	60.74	56.24	58.99	55.74
156	68.15	65.93	58.97	56.09
174	77.04	73.38	59.77	57.29
194	78.52	74.39	54.64	50.96
231	82.96	78.94	48.48	44.86
241	83.70	80.67	46.89	43.62
251	85.19	81.07	45.82	42.13
260	85.19	81.07	44.23	41.47
269	86.67	82.91	43.49	41.15

Table 6.13: Results for Embedded Systems

keywords	Recall(%)		Precision(%)	
	R_w	R_r	P_w	P_r
96	48.25	44.47	71.88	67.07
135	58.74	55.09	62.22	59.68
152	62.94	59.23	59.21	56.09
170	68.53	64.84	57.65	54.16
190	70.63	67.04	53.16	50.56
226	71.33	68.95	45.13	41.62
235	74.83	71.36	45.13	41.61
245	77.62	74.36	45.31	41.79
253	80.42	77.96	45.45	41.92
262	81.82	78.28	44.66	40.46

Table 6.14: Results for Numerical Analysis

keywords	Recall(%)		Precision(%)	
	R_w	R_r	P_w	P_r
99	61.29	57.69	38.38	34.18
138	64.52	61.08	28.99	26.74
156	67.74	64.86	26.92	25.86
174	80.65	76.68	28.74	25.04
194	82.26	78.28	26.29	22.91
232	91.94	89.81	24.57	21.78
242	95.16	92.05	24.38	21.61
250	98.39	94.46	24.40	21.67
255	98.39	94.46	23.92	20.03
263	98.39	94.46	23.19	19.75

Table 6.15: Results for System Analysis And Design

keywords	Recall(%)		Precision(%)	
	R_w	R_r	P_w	P_r
98	45.38	41.52	55.10	51.28
138	48.74	44.10	42.03	39.93
158	53.78	50.30	40.51	37.41
177	59.66	56.82	40.11	37.25
197	59.66	56.82	36.04	32.53
236	62.18	59.66	31.36	27.58
246	64.71	61.03	31.30	27.49
256	67.23	63.25	31.25	27.41
266	70.59	65.52	31.58	27.80
276	70.59	65.52	30.43	26.27

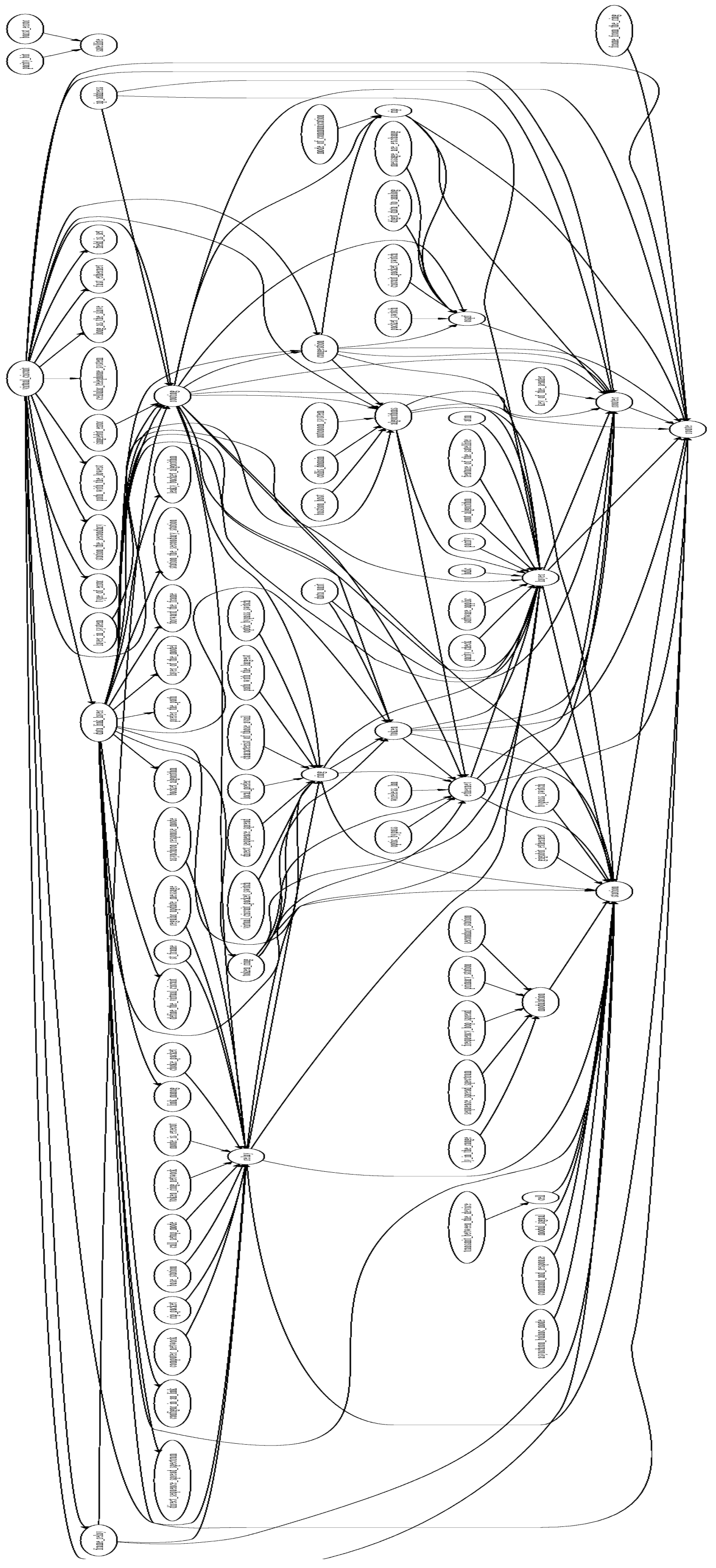


Figure 6.8: DAG for Computer Networks by System-2

6.5 System-1 Vs. System-2

Table - 6.16 and 6.17 depicts the confusion metrics for the best results of system-1, and system-2, respectively.

Table 6.16: Confusion Matrix for System - 1

Subject	**System Generated**		**Expert identified**		**True Positive**		**False Positive**		**False Negative**	
	Keywords	Relations	Keywords	Relations	Keywords	Relations	Keywords	Relations	Keywords	Relations
Software	131	224	124	211	80	121	51	103	44	90
Networks	136	250	172	323	81	137	55	113	91	186
Cryptography	139	322	135	313	79	173	60	149	56	140
Embedded	136	296	143	313	71	144	65	152	72	169
Numerical	131	228	62	95	42	61	89	167	20	34
Operating	125	286	107	237	61	126	64	160	46	111
SAD	138	193	119	163	62	79	76	114	57	84

Table 6.17: Confusion Matrix for System - 2

Subject	**System Generated**		**Expert identified**		**True Positive**		**False Positive**		**False Negative**	
	Keywords	Relations	Keywords	Relations	Keywords	Relations	Keywords	Relations	Keywords	Relations
Software	95	160	124	211	74	117	21	43	50	94
Networks	97	173	172	323	79	136	18	37	93	187
Cryptography	100	228	135	313	74	160	26	68	61	153
Embedded	96	208	143	313	69	139	27	69	74	174
Numerical	99	160	62	95	38	55	61	105	24	40
Operating	98	195	107	237	45	81	53	114	62	156
SAD	98	132	119	163	54	68	44	64	65	95

From the above tables, it is clear that *System-1* generated more number of `False Positives` than *System-2*. This is because *System-2* gives good performance at lower number of keywords whereas *System-1* gives same performance at hight number of keywords. As we have to generate more keywords checking relation between more keywords takes more time.

The processing time of *System-2* is lesser than *System-1*. This is because, stemming decreases the total number of unique words in the given corpora. Legitimate keywords also got good `tfidf score` due to stemming, as stemming increased the count of the root words, `tfidf score` increased.

As person, place or an organization do not have any pre-requisites or follow-up, removing them through named entity recognition and elimination helped us in finding the legitimate keywords in *System-2*.

6.6 Observations and Interpretations

After conducting different experiments, we observed that `Recall` was maximum when the number of keywords was nearly equivalent to 130. The optimum value of `Recall` was obtained when the number of unigrams was 50, number of bigrams was 40, number of trigrams was 30, and number of fourgrams was 20. This is because most of the keywords in any subject are unigrams, there are very few fourgrams, and there may be hardly any fivegrams. However, more experiments with lecture notes in various subjects are required before these results can be generalized. Moreover, the goodness metric can be improved by considering a higher number of keywords, at the cost of increase in execution time.

To calculate `precision` and `recall`, keyword/relation must be either correct or incorrect. But most of the times keywords/relations may be somewhat relevant or somewhat irrelevant. Others may be very relevant and others completely irrelevant. This problem is complicated by individual perception: what is correct keywords/relations to one person may not be correct to another.

Measuring `recall` is difficult because it is often difficult to identify the correct keywords for a given subject, It completely depends on the individual perspective as discussed above. So `recall` can be identified by getting the correct keywords from more than one experts against getting the keywords from single expert.

Measuring `True Negative` is difficult as neither our system, nor expert gives the list of wrong keywords. So in confusion matrix we take only three cells and ignore True Negative.

Chapter 7

Conclusion & Future Work

Here we presented a completely automatic ontology generator whose performance is fairly good. We have observed some conflicts in the system and the expert answers in the sense that some relations which were generated by our system is not valid. and some important results which should present were not generated. Keywords found are fairly good, overall system performance, `recall` can be increased by taking top 200 keywords instead of taking top 130 keywords by compromising time complexity.

Currently many methodologies, tools and languages are available for building ontologies. Too many ways will mislead the Ontology developers and users, also it will create difficulties in integrating the existing Ontologies. A single centralized platform will help developers and the users of ontology to understand better and integration also can be made easy. The future work in this field should be done towards the creation of a common platform for ontology developers to facilitate ontology development, exchange, evaluation, evolution and management.

Appendix A

Stop Words

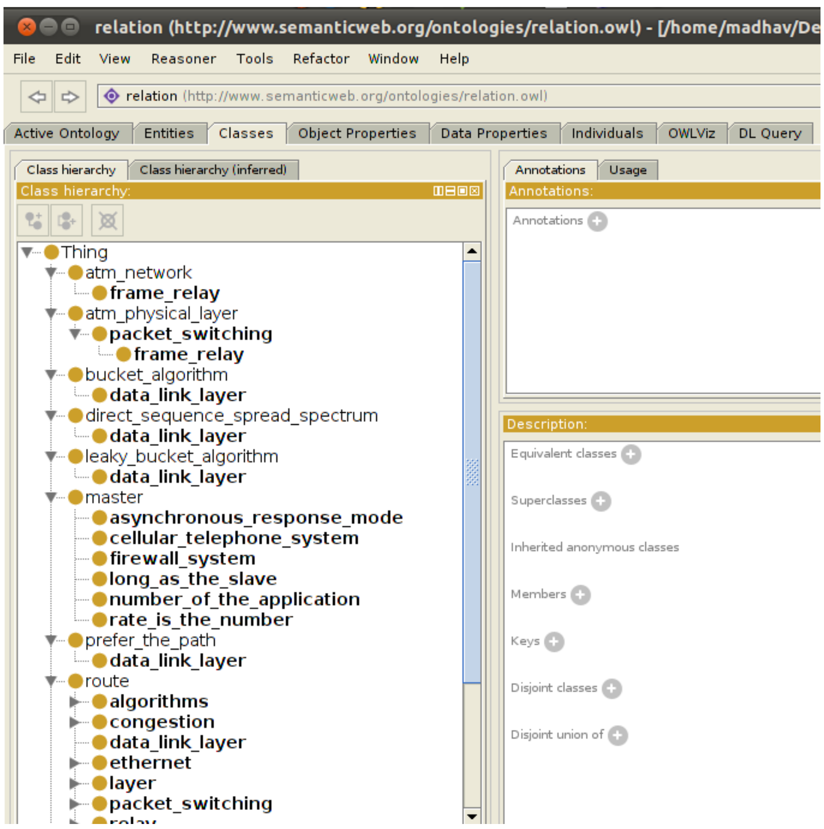
Following is the list of common words which are used frequently in any text. These filtered words are known as “Stop words”. Our searching technique doesn’t consider the following words in the process of constructing ontology as well as in searching.

a, able, about, above, abst, accordance, according, accordingly, across, act, actually, added, adj, adopted, affected, affecting, affects, after, afterwards, again, against, ah, all, almost, alone, along, already, also, although, always, am, among, amongst, an, and, announce, another, any, anybody, anyhow, anymore, anyone, anything, anyway, anyways, anywhere, apparently, approximately, are, aren, arent, arise, around, as, aside, ask, asking, at, auth, available, away, awfully, b, back, be, became, because, become, becomes, becoming, been, before, beforehand, begin, beginning, beginnings, begins, behind, being, believe, below, beside, besides, between, beyond, biol, both, brief, briefly, but, by, c, ca, came, can, cannot, can’t, cause, causes, certain, certainly, co, com, come, comes, contain, containing, contains, could, couldnt, d, date, did, didn’t, different, do, does, doesn’t, doing, done, don’t, down, downwards, due, during, e, each, ed, edu, effect, eg, eight, eighty, either, else, elsewhere, end, ending, enough, especially, et, et-al, etc, even, ever, every, everybody, everyone, everything, everywhere, ex, except, f, far, few, ff, fifth, first, five, fix, followed, following, follows, for, former, formerly, forth, found, four, from, further, furthermore, g, gave, get, gets, getting, give, given, gives, giving, go, goes, gone, got, gotten, h, had, happens, hardly, has, hasn’t, have, haven’t, having, he, hed, hence, her, here, hereafter, hereby, herein, heres, hereupon, hers, herself, hes, hi, hid, him, himself, his, hither, home, how, howbeit, however, hundred, i, id, ie, if, i’ll, im, immediate, immediately, importance, important, in, inc, indeed, index, information, instead, into, invention, inward, is, isn’t, it, itd, it’ll, its, itself, i’ve, j, just, k, keep, keeps, kept, keys, kg, km, know, known, knows, l, largely, last, lately, later, latter, latterly, least, less, lest, let, lets, like, liked, likely, line, little, ’ll, look, looking, office, figure, version, substitution, attribute, kharagpur, looks, ltd, m, made, mainly, make, makes, many, may, maybe, me, mean, means, meantime, meanwhile, merely, mg, might, million, miss, ml, more, moreover, most, mostly, mr, mrs, much, mug, must, my, myself, n, na, name, namely, nay, nd, near, nearly, necessarily, necessary, need, needs, neither, never, nevertheless, new, next, nine, ninety, no, nobody, non, none, nonetheless, noone, iit, cse, nor, normally, nos, not, noted, nothing, now, nowhere, o, obtain, obtained, obviously, of, off, often, oh, ok, okay, old, omitted, on, once, one, ones, only, onto, or, ord, other, others, otherwise, ought, our, ours, ourselves, out, outside, over, overall, owing, own, p, page, pages, part, particular, particularly, past, per, perhaps, placed, please, plus, poorly, possible, possibly, potentially, pp, predominantly, present, example, previously, primarily, probably, promptly, proud, provides, put, q, que, quickly, quite, qv, r, ran, rather, rd, re, readily, really, recent, recently, ref, refs, regarding, regardless, regards, related, relatively, research, respectively, resulted, resulting, results, right, run, s, said, same, saw, say, saying, says, sec, section, see, seeing, seem, seemed, seeming, seems, seen, self, selves, sent, seven, several, shall, she, shed, she'll, shes, should, shouldn't, show, showed, shown, showns, shows, significant, significantly, similar, similarly, since, six, slightly, so, some, somebody, somehow, someone, somethan, something, sometime, sometimes, somewhat, somewhere, soon, sorry, specifically, specified, specify, specifying, state, states, still, stop, strongly, sub, substantially, successfully, such, sufficiently, suggest, sup, sure, t, take, takes, taken, taking, tell, tends, th, than, thank, thanks, thanx, that, that'll, thats, that've, the, their, theirs, them, themselves, then, thence, there, thereafter, thereby, thered, therefore, therein, there'll, thereof, therere, theres, thereto, thereupon, there've, these, they, theyd, they'll, theyre, they've, think, this, those, thou, though, thoughh, thousand, throug, through, throughout, thru, thus, til, tip, to, together, too, took, toward, towards, tried, tries, truly, try, trying, ts, twice, two, u, un, under, unfortunately, unless, unlike, unlikely, until, unto, up, upon, ups, us, use, used, useful, usefully, usefulness, uses, using, usually, v, value, various, 've, very, via, viz, vol, vols, vs, w, want, wants, was, wasn't, way, we, wed, welcome, we'll, went, were, weren't, we've, what, whatever, what'll, whats, when, whence, whenever, where, whereafter, whereas, whereby, wherein, wheres, whereupon, wherever, whether, which, while, whim, whither, who, whod, whoever, whole, who'll, whom, whomever, whos, whose, why, widely, will, willing, wish, with, within, without, won't, words, world, would, wouldn't, www, x, y, yes, yet, you, youd, you'll, your, youre, yours, yourself, yourselves, you've, z, zero

Appendix B

Other Results

Protégé out put for computer networks is shown in the figure - B.1.



relation (http://www.semanticweb.org/ontologies/relation.owl) - [/home/madhav/De
File Edit View Reasoner Tools Refactor Window Help
relation (http://www.semanticweb.org/ontologies/relation.owl)
Active Ontology
Entities
Classes
Object Properties
Data Properties
Individuals
OWLViz
DL Query
Class hierarchy
Class hierarchy (inferred)
Class hierarchy:
Annotations
Usage
Annotations:
Annotations
Description:
Equivalent classes
Superclasses
Inherited anonymous classes
Members
Keys
Disjoint classes
Disjoint union of

```
Thing
  atm_network
    frame_relay
  atm_physical_layer
    packet_switching
      frame_relay
  bucket_algorithm
    data_link_layer
  direct_sequence_spread_spectrum
    data_link_layer
  leaky_bucket_algorithm
    data_link_layer
  master
    asynchronous_response_mode
    cellular_telephone_system
    firewall_system
    long_as_the_slave
    number_of_the_application
    rate_is_the_number
  prefer_the_path
    data_link_layer
  route
    algorithms
    congestion
    data_link_layer
    ethernet
    layer
    packet_switching
```

Figure B.1: Computer Networks ontology developed by our system using protégé

In the Evaluation chapter, we have shown the experiments for "Computer Networks" course only. The other courses for which we have also done experiments for are "Operating System","Artificial Intelligence", "Embedded Systems", "Software Engineering", "System Analysis and Design", "Embedded Systems", "Cryptography" and "Numerical Analysis". The final ontology in the form of Directed Acyclic graph is shown in the following figures. Here ellipses (Nodes) represents the keywords, whereas Arrows (links) represent the relationship.

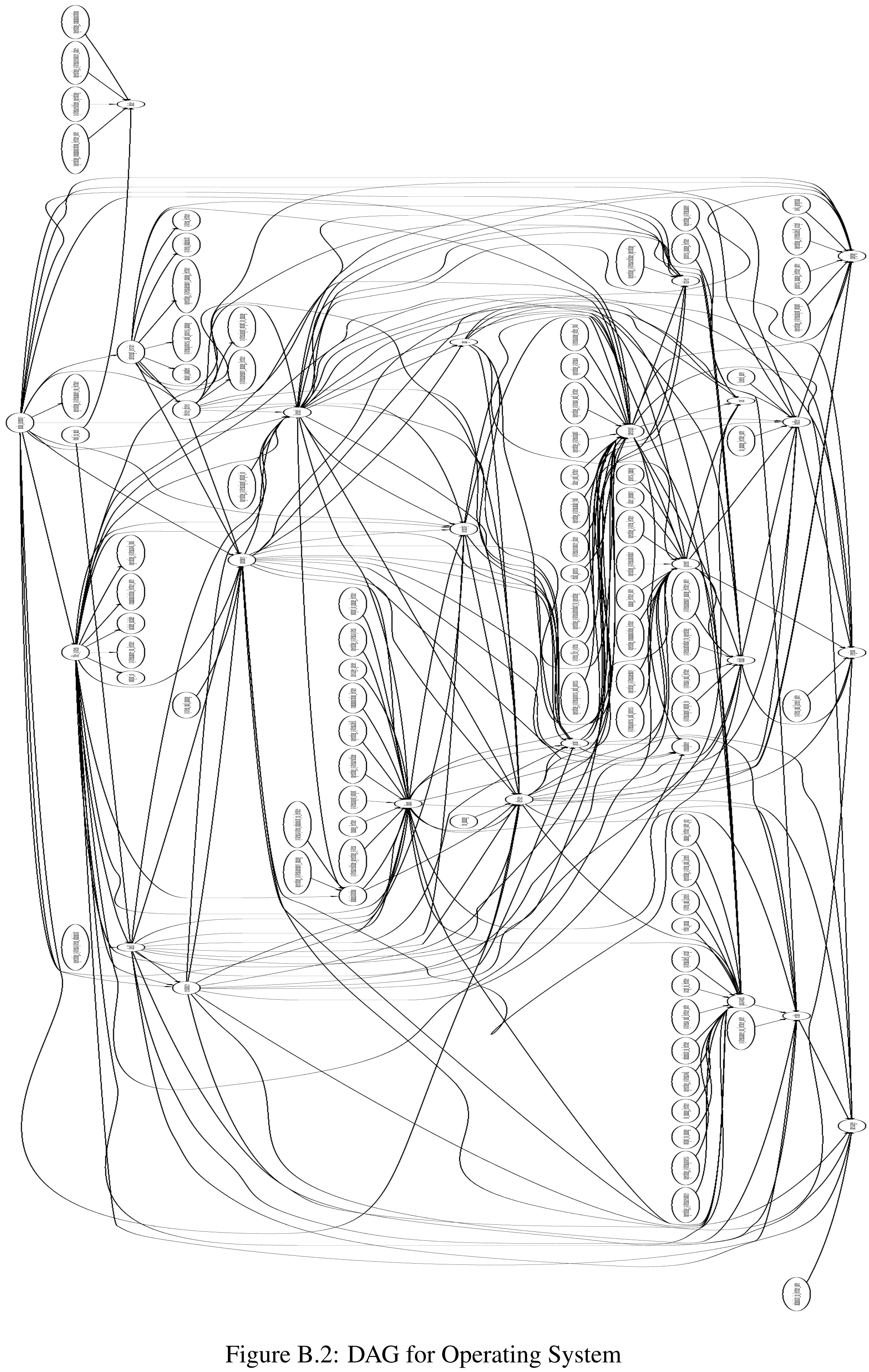


Figure B.2: DAG for Operating System

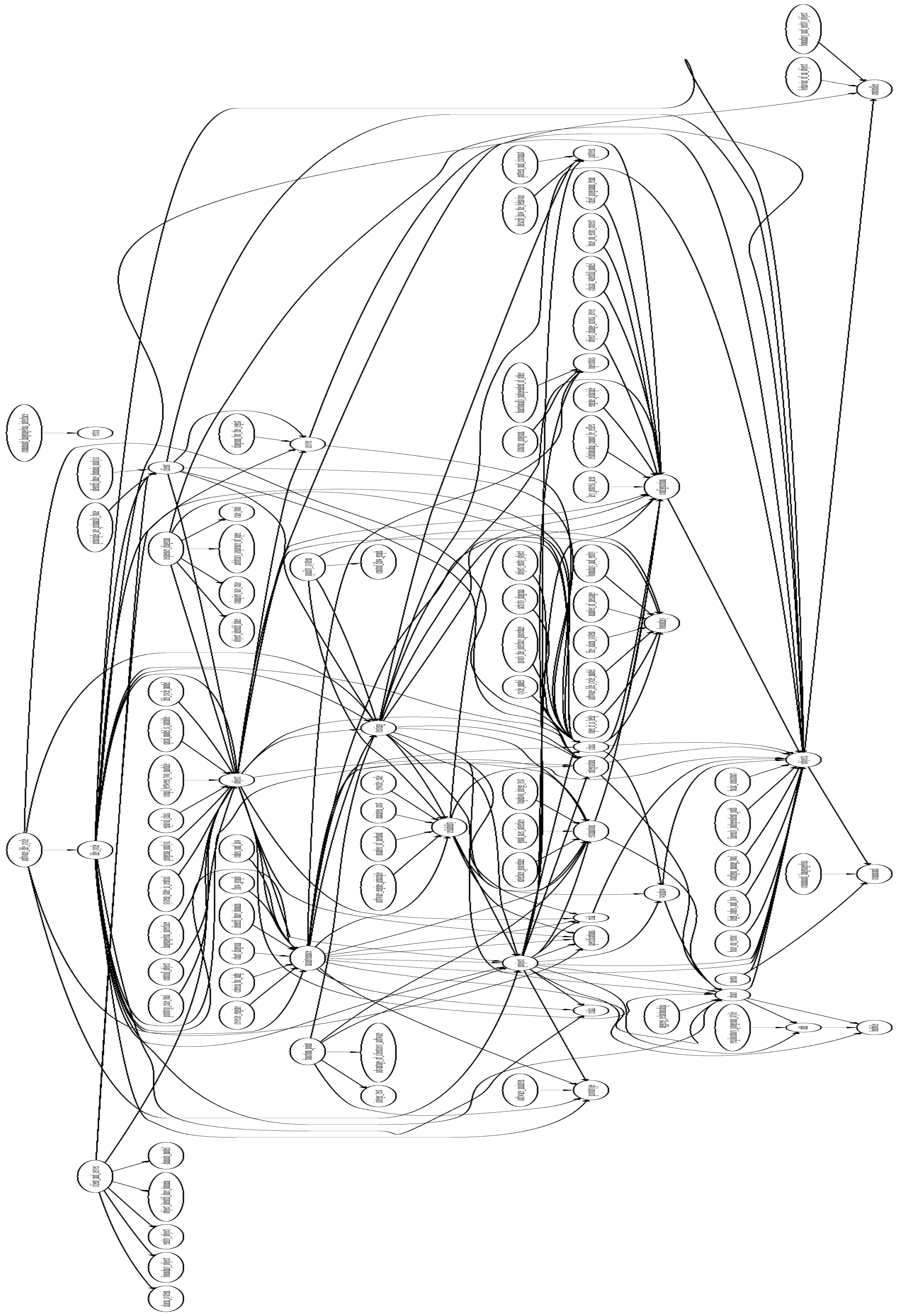


Figure B.3: DAG for Software Engg

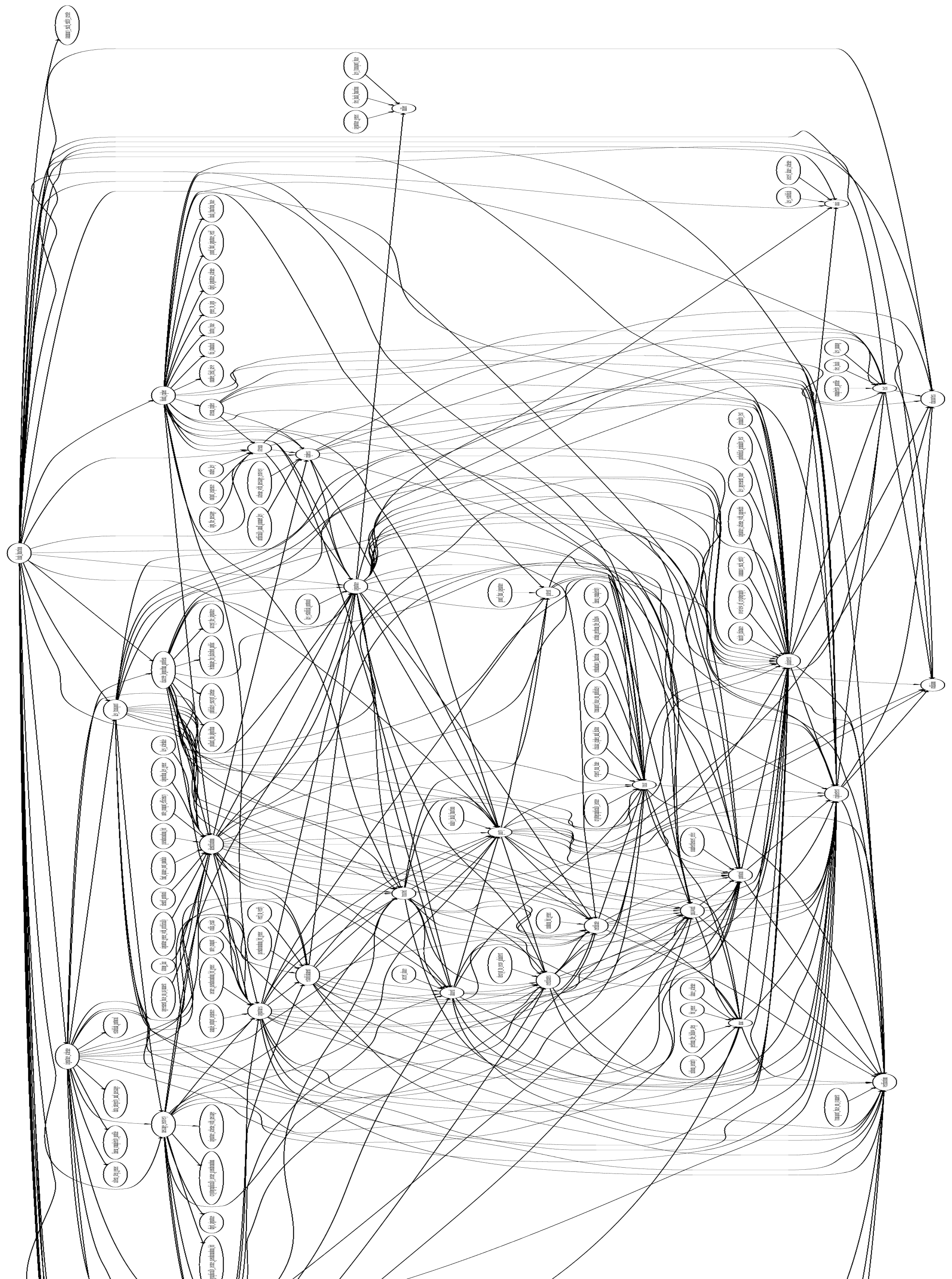


Figure B.4: DAG for Cryptography

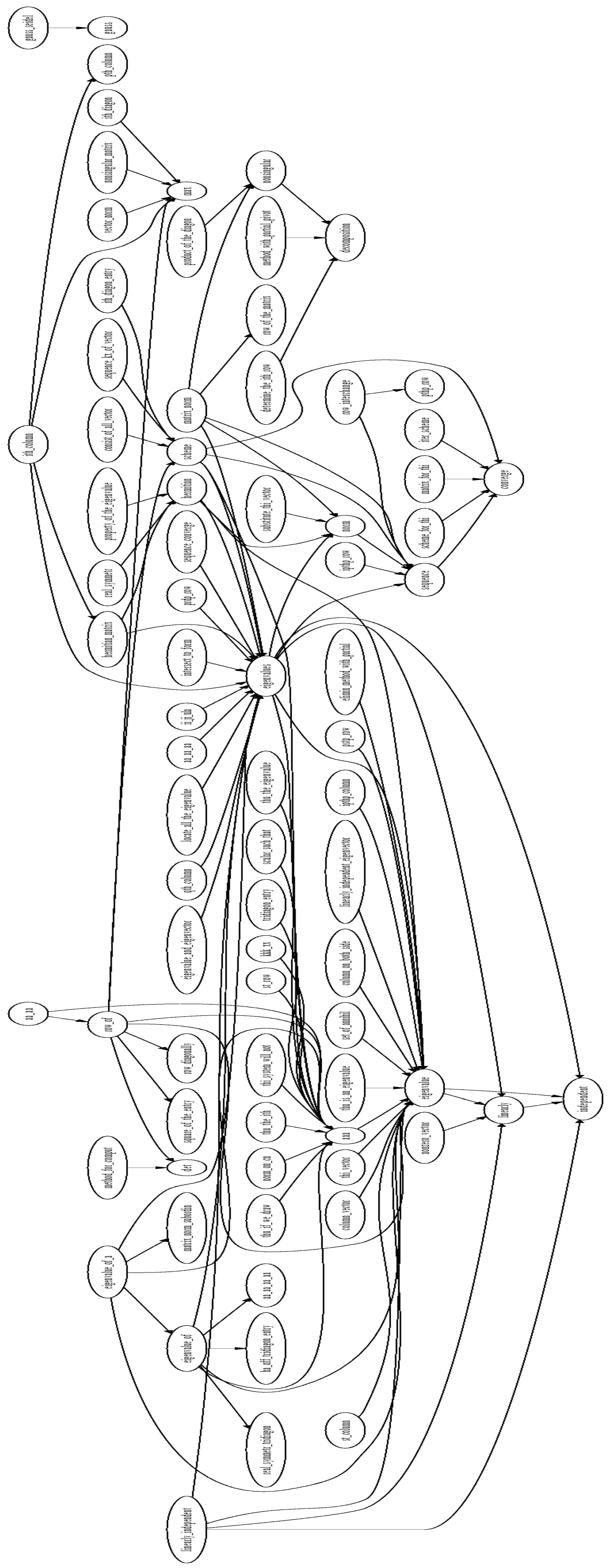


Figure B.5: DAG for Numerical Analysis

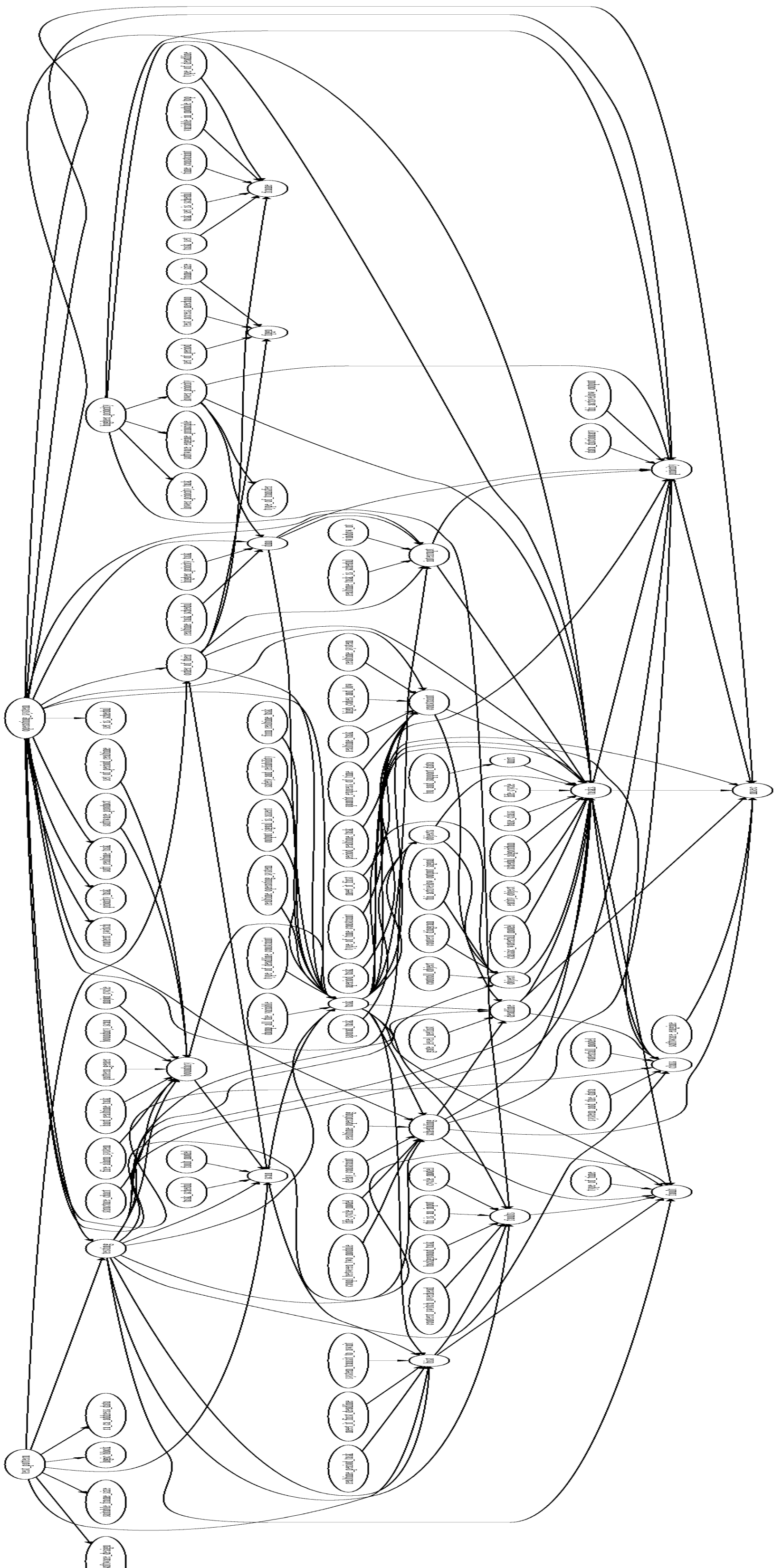


Figure B.6: DAG for Embedded System

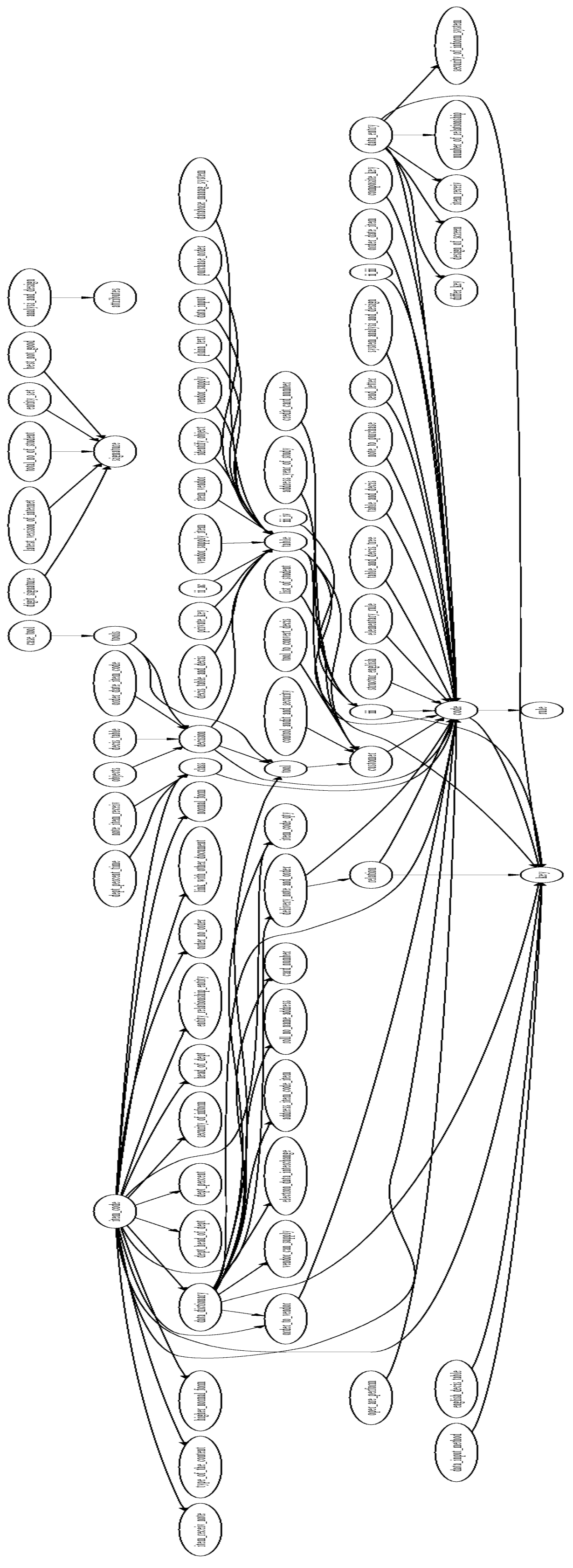


Figure B.7: DAG for System Analysis and Design

Bibliography

[APR94] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[Cmap08] J. D. Novak and A. J. Ca nas. The theory underlying concept maps and how to construct and use them. In *Technical Report IHMC CmapTools 2006-01 Rev 01-2008*. Florida Institute for Human and Machine Cognition, 2008.

[DOT] Emden Gansner, Eleftherios Koutsofios, and Stephen North. Drawing graphs with dot, Jan 2006.

[Fern03] Óscar Corcho, Mariano Fernández-lópez, and AsunciÓn GÓmez-pérez. Methodologies, tools and languages for building ontologies: Where is their meeting point. *Data & Knowledge Engineering*, 46:41–64, 2003.

[GAV05] Tatiana Gavrilova, Rosta Farzan, and Peter Brusilovsky. One practical algorithm of creating teaching ontologies. 2005.

[Grub95] Thomas R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. *Int. J. Hum.-Comput. Stud.*, 43:907–928, December 1995.

[Grun95] Michael Gruninger and Mark S. Fox. Methodology for the design and evaluation of ontologies. In *Workshop on Basic Ontological Issues in Knowledge Sharing*. IJCAI-95, 1995.

[Ivan07] Ivan Bedini and Benjamin Nguyen. Automatic ontology generation: State of the art. In *PRiSM Laboratory Technical Report*. University of Versailles, 2007.

[Khan02] Latifur Khan and Feng Luo. Ontology construction for information selection. In *Proceedings of the 14th IEEE International Conference on Tools with Artificial Intelligence*, ICTAI '02, pages 122–, Washington, DC, USA, 2002. IEEE Computer Society.

[LIU04] DA-YOU LIU. Learning owl ontologies from free texts. In *Machine Learning and Cybernetics*, volume 2, pages 1233 –1237, 2004.

[NER05] Jenny Rose Finkel, Trond Grenager, and Christopher Manning. Incorporating non-local information into information extraction systems by gibbs sampling. In *In ACL*, pages 363–370, 2005.

[NOY01] Natalya F. Noy and Deborah L. McGuinness. Ontology development 101: A guide to creating your first ontology. Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report, March 2001.

[OIL] Dieter Fensel, Frank Van Harmelen, Ian Horrocks, Deborah L. Mcguinness, and Peter F. Patel-schneider. Oil: An ontology infrastructure for the semantic web. *IEEE Expert / IEEE Intelligent Systems*, 16:38–45.

[OWL] D. L. Mcguinness and F. Van Harmelen. Owl web ontology language overview. *World Wide Web*, 2004.

[PDFB] Apache. The apache software foundation — pdfbox. Accessed 16-February-2011.

[RAM03] Juan Ramos. Using tf-idf to determine word relevance in document queries. *First International Conference on. Machine Learning*, 2003.

[RDF] Pierre-Antoine Champin June. Rdf tutorial, April 2001.

[SALT02] Deryle Lonsdale, Yihong Ding, David W. Embley, and Alan Melby. Peppering knowledge sources with salt: Boosting conceptual content for ontology generation. In *AAAI Workshop for Semantic Web Meets Language Resources, The Eighteenth National Conference on Artificial Intelligence*, pages 30–36. AAAI Press, 2002.

[Stem80] Martin Porter. An algorithm for suffix stripping. In *Workshop on Multimedia Information Systems*, 1980.

[Term99] Brigitte Biebow and Sylvie Szulman. Terminae: A linguistic-based tool for the building of a domain ontology. In *Knowledge Acquisition, Modeling and Management*, pages 49–66, 1999.

[Topia] Python. package index — topia.termextract 1.1.0. Accessed 16-February-2011.

[WikiLuc] Wikipedia. Lucene — wikipedia, the free encyclopedia, 2011. [Online; accessed 16-February-2011].

[Wikidep] Wikipedia. Dependency graph — wikipedia, the free encyclopedia, 2011. [Online; accessed 16-February-2011].

[Wikikey] Wikipedia. Terminology extraction — wikipedia, the free encyclopedia, 2011. [Online; accessed 16-February-2011].

[Wikirdfs] Wikipedia. Rdf schema — wikipedia, the free encyclopedia, 2010. [Online; accessed 15-June-2011].

[Wikistem] Wikipedia. Stemming — wikipedia, the free encyclopedia, 2011. [Online; accessed 27-May-2011].

[XML] Wikipedia. Xml — wikipedia, the free encyclopedia, 2011. [Online; accessed 15-June-2011].

[XMLS] Michel Klein, Dieter Fensel, Frank van Harmelen, and Ian Horrocks. The relation between ontologies and xml schemas. *Electronic Transactions on Artificial Intelligence*, 2001.

[YUN09] Yun Hong-yan, Xu Jian-liang, Wei Mo-ji, and Xiong Jing. Development of domain ontology for e-learning course. In *IT in Medicine Education, 2009. ITIME '09. IEEE International Symposium on*, 2009.