Construction and Maintenance of a Web Warehouse

Benjamin Nguyen

December 26, 2003

Contents

Acknowledgements

There are of course countless people I would like to thank, and it would take me many pages to express my deepest gratitude to all the people I have come to meet during these last three years. Needless to say that my thankfulness extends much further than these simple words of thanks.

I would like to thank Serge Abiteboul, my advisor, for accepting me in his research team, and for the great discussions we have had. Working with you Serge has been a wonderfull and enriching experience, due to your great knowledge, wisdom, acuity and of course, kindness and humour... that maybe I should have spelt the American way.

I would like to thank all the other reasearchers of the Verso/Gemo team, a great place to work in because of all the knowledgeable yet ever so friendly people I met along the way. I would like to extend my thoughts more specifically to Gregory, Antonella, Jerome, Laurent, Mihai, Jeremy and Kjetil with whom I had a closer collaboration.

I would like to thank Michalis Vazirgiannis, and all the members of the DB-Net team of the Athens University of Economics and Business, for their welcome, and Michel Scholl for providing me the opportunity of these visits. During all my stays in Greece, you always made me feel at home, and this also managed to create an intense research activity. Special thanks to all the students at AUEB, and especially Maria and Iraklis, whom I worked with, and went out with. I really hope visiting you soon.

I would also like to thank the members of the IASI group, of the LRI, for the scientific advice on many matters, and in particular Brigitte Safar.

Finally, I would like to thank Mireille Tadjeddine and Alain Petit, of the ENS-Cachan, for pointing me in the right direction, all those years ago.

Chapter 1

Introduction

During the last five years, the size of the World Wide Web has grown from an already amazing billion pages, to several billions (according to the statistics of Google [70] or the Internet Archive [18]). The Web has brought a deep change to the way humans apprehend information: most of the things you know are out there, and most of what you want to know can also be found there. A major breakthrough that led to this new situation is the development of large scale Web search engines. Moreover, this immediate access to large quantities of often *evolving* information has created new needs and new problems; the need for more support to discover the information, the need to analyze and process it, and the need to store it and query it. In this work, we take a closer look at large systems that obtain and process data from the Web, and organize it into what is called a *Web Warehouse*.

Data Warehousing, i.e., the processing of large quantities of data, and the development of specific methods to do so, has been the topic of past work, such as [155]. Our general problem here is similar, but turned towards documents and the Web. The information is no longer sitting in relational databases, very structured and ready to be processed. Information needs to be discovered, retrieved, analyzed, stored, enriched and constantly monitored for changes. In this thesis, we propose to analyze how such systems should be designed, based on experiences with two Web Warehousing applications, Xyleme and Thesus. We study novel features that are specific to large scale, changing Web Warehouses. The Xyleme project considered the problem of retrieving XML information from the Web, storing, querying and monitoring it. The Thesus project addressed the issues of constructing and using thematic collections of HTML pages. Both these experiments led us to articulate the way Web Warehouses should be designed, work that took place within the Spin project.

The original goal of the Xyleme project was to create a large scale XML

warehouse, that would contain all the XML data from the Web. The tasks to achieve ranged from the crawling of the Web, to the native XML storage and the processing of the pages discovered. We participated in the project by developing the monitoring system for Xyleme. The role of the monitoring module was to detect the changes that would take place in Web pages, and alert systems or users of these changes. The algorithms and the system were developed bearing scalability in mind; the system can process several million documents a day. It is currently in production at Xyleme S.A., a company that started on the research in the Xyleme Project.

The Thesus system was developed in a cooperation between the Verso/Gemo group at INRIA and the database group at Athens University of Economics and Business. The goal of the system was to construct warehouses of Web data, enriching and clustering the documents into thematic subsets. We participated in the general design of the system. We also worked more specifically on the semantic enhancement of documents and their clustering, using information extracted from links, and ontologies. A prototype system is running, and individual modules have been packaged as Web services.

These two experiences led to a better understanding of the specific needs, in terms of architecture, specification language, and tools for Web based, theme oriented warehouses. Building on them, the Spin (Set of Pages of Interest) project proposes a general framework, based on a declarative specification language that can be used to design a thematic Web Warehouse. This is based on interactions with Web Services performing the necessary tasks for obtaining and enriching the content of the Warehouse. The administrator of a Thematic Web Warehouse designs both the content structure, and the interactions with services. The Spin prototype has been implemented using the Active XML language and system [3, 2].

The thesis is divided into three parts. The first part, Chapter 3, discusses the architecture of the Xyleme and Thesus systems, and proposes the Spin approach for modeling a Web Warehouse. Then we discuss in more detail tow important modules of such systems namely in Chapter 4 the Xyleme Monitoring System, and in Chapter 5 the thematic enhancement and clustering in Thesus.

Chapter 2

State of the Art

In this thesis, we study the following problem : *how can we define and construct a Web Warehouse , i.e., using information found on the Web.* Our general context is the Web, thus we will give some precisions on its specificities, and the studies surrounding our object. We will give some precisions on work regarding Web Warehousing, and related work on the more specific modules that are also the topic of this thesis, the monitoring of events in a Web Warehouse, and the semantic enrichment and clustering of the documents it contains.

2.1 The World Wide Web

The World Wide Web from the database point of view

The Web constitutes the largest body of information accessible to any individual throughout the history of humanity and keeps growing at a healthy pace [18]. The Web in itself is central to some recent work : Its structure for instance [32], or its diameter [14] are some of the many questions that it has raised.

An important characteristic of the Web is that most of it is unstructured, or at least very loosely structured, and quite heterogeneous. A major evolution is occurring that will dramatically simplify the task of developing applications with this data. The coming of XML [4, 152, 151, 102] has proven to be less than a revolution than what was expected, but nonetheless, semi-structured data has made its way onto the visible Web [105], and there are a growing number of applications using XML and XSL for publishing data on the Web. What was once only a huge collection of documents is turning into a massive database. Let us stress at this point, that although there is ongoing

research regarding what is called the *hidden Web*[125], page unreachable by hyperlinks, or pages hidden behind Web forms that need prior registration in order to provide significant information, we restrict our representation of the Web to pages that have at least one incoming or outgoing link, as described in [32], and throughout this thesis when we refer to pages on the World Wide Web, we mean pages that are accessible on the World Wide Web by a static URL.

Indeed, the Web has become a new topic of the Database world. For a long time already, researchers have been looking at better ways of querying the Web [99, 9], and [67] propose an interesting survey of what database techniques can bring to the Web. These techniques can be divided into three points according to Florescu *et.al.*:

- Modeling and querying the Web, using for instance an SQL like language [21] such as the WebSQL project. The novelty is the inclusion of topology based queries that take into account the link structure of the Web.
- Web site construction and restructuring. An interesting application would be for instance the WebML [154] application, which helps users construct their Web site using typical database modeling techniques.
- Information extraction and integration, which is the theme this thesis sets itself in.

2.1.1 Searching the World Wide Web

Search Engines It is impossible to picture today the Web without Search engines [97], that have been around since the 1990's [90]. They provide an intuitive solution to retrieve information from the Web, and the quality of their results has greatly improved since the introduction of Google[30, 70], and algorithms such as PageRank[118]. The major breakthrough that was achieved by Google was to provide an accurate means of measuring the relative importance of a page on the Web, and by assuming that the importance of a page is proportional to it's absolute relevance.

The PageRank model is based on weight normalization and models web surfing based on a random walk model. Although the model may not be very fitting to the way real users browse the Web, it does prove accurate with regards to user satisfaction [110]. In order to compute each page's importance, the PageRank algorithm uses the adjacency matrix of the pages[1]. More recent algorithms, such as [6, 8] do not need such information.

[1] $M = [m_{i,j}]$ is the adjacency matrix, a symmetric $N \times N$ matrix where N is the number

Another measure of page importance is HITS[2], in which importance is a function of mutual reinforcement between Hub and Authority pages [83]. [55] try to combine both approaches into a unified framework. In this thesis, we will extend the concepts of Hub and Authorities to *semantic* Hubs and Authorities.

Information Retrieval Techniques on the Web Information Retrieval (IR) is a well established domain [131] that now has a new field of application : Web based IR. [23] isolate four different approaches of this problem in the Web context:

- human or manual indexing
- automatic indexing, using classical IR techniques
- intelligent or agent based indexing, such as using Crawlers and Robots
- meta-data, RDF[3] and annotation-based indexing.

[88] propose a survey of these approaches, along with user interface considerations, but these are far beyond the scope of this thesis.

Crawlers Central to the task of retrieving information from the Web are the Crawlers. They started of as simply programs that tried to scan the Web and find all the pages that were to be discovered. But rapidly, with the growth of the Web, the task of the Crawlers became more different, and maybe more difficult : they were no longer capable of finding and fetching [4] all the pages from the Web, but they had to adopt a strategy in order to retrieve the *best* pages from the Web. As we have seen earlier, by using ranking algorithms, pages can be given an absolute importance ranking. [44] study the roots of "intelligent" crawling. Which pages (important ones) should be visited first ? They propose a certain number of tools, such as a metric in order to evaluate their proposals, and have implemented a crawler based on these ideas.

We also mention here orthogonal improvements to Crawlers, such as incremental crawling[42] or parallel crawling [43]. Both these aspects were central to the XyRo [104] which was used in the Xyleme system.

---

of (indexed) pages on the Web containing the value $m_{i,j} = 1$ if page i points to page j and 0 if this is not the case.

[2]Hypertext Induced Topic Selection

[3]Resource Description Framework [146]

[4]fetching a page means downloading the page contents

On of the latest evolutions of Crawlers is *focused crawling* [36]. The goal is no longer to fetch all kinds of pages, but to restrict the pages retrieved to a certain type, most often pages on a given topic, or pages that have similar structure. The goal of course is to be more efficient than gathering a lot of documents, indexing them then querying them. The Xyleme system detailed in this thesis does not use focused crawling, but the Thesus system, selective in the pages retrieved, can be seen as some sort of focused crawler. In our latest project, *e.dot*[59], focused crawling is taken fully into account [101].

2.1.2 The Semantic Web

Meta-data Meta-data, from the Greek prefix *meta*, meaning *among, together, with*, is additional data which gives extra information on the document being processed. One can encounter meta-data on the web, this information is invisible to the user browsing the document, and would be information such as the last date of access of the page, the format of the document, etc. Some initial efforts such as [100, 89] aim at attaching a set of tags, such as *author, title, type* with the corresponding information to each web document. Naturally, these tags need to be constructed by the author of the Web document. However, these standards do not seem to have yet been adopted by the general public. For a more precise survey on meta-data, we refer to [23]. The main problem of these meta-tags is that the information is specific to each document on the Web. Thus the management of collections of documents is not easy. There are many problems involving the update and merging of the meta-data, and its application to collections. In order to solve these problems, the *Resource Description Framework (RDF)* [146] was introduced, along with means of querying such documents [86].

Meta information can also be important in the prospect of crawlers : [29] propose web servers export specific meta-data archives describing their content. The authors claim that this will significantly reduce the bandwidth when crawling Web sites. [96] is also a strong advocate for the use of RDF in order to enhance the quality of search results on the Web.

Semantics In order to offer better processing of information, a unified representation for web resources (data and services) is becoming a necessity. The use of ontologies to provide the means to machines of understanding the data they are manipulating is increasing. In our work we only use the simplest of ontologies [64], a taxonomy in fact, as per the following definition:

Definition 2.1.1 *In this thesis, we call ontology a tree containing concepts liked between each other by a is-a relation.*

With the emergence of this Semantic Web [140], the study of ontologies and their uses has increased, since they provide a shared and common understanding of a domain that can be communicated between people and application systems

There are many sub problems within the Semantic Web, one of the most important one is the mapping between ontologies [56], the construction of ontologies [63].

2.2 Web Warehousing

There has already been a lot of work on the topics of data warehousing, mediation and integration of data [155].

We refer to [38, 142, 37] for a specific survey on OLAP, Data Warehousing and Materialized views, and a number of methodologies and tools for constructing classical data warehouses. To our knowledge, no methodology exists for defining data warehouses based on Web resources. Some tools do exist that facilitate some aspects of this task, e.g., Jalios [82], which is a tool to build web portals.

Our work does not address directly the various functionalities of data warehousing such as the well known Data Cubes [72]. It is primarily a high-level method to define how to use such functionalities provided as Web services. The basis for our work is XML storage and querying [11, 150, 149]), as well as Web services [147, 148]. The main novelty of this part of the thesis lies in the use of a data model combining an XML description of data and meta-data and Web-service for processing the data and meta-data.

2.3 Temporal Evolution in a Warehouse

2.3.1 Monitoring Changes

Web users are often concerned by changes in pages they are interested in. The present work has been developed in the context of Xyleme [162], a dynamic XML warehouse for the Web. We monitor the flow of documents that are discovered and refreshed by the system. The monitoring of a large database is a typical warehousing problems [155]. However, a basic distinction is that since we do not control the (external) Web sites, we have to detect changes at the time we are fetching the pages. For HTML pages, we have their signature and we can only detect whether they have changed or not. For XML, we also monitor changes at the element level e.g., a new product has been introduced in a catalog.

2.3.2 Versioning Data

A lot of works in databases involving time is about versioning data [40, 39]. Although we do have a versioning mechanism in the Xyleme system, this is not the topic of this thesis. For more information regarding this topic, we refer to Gregory Cobena's work [45] Temporal query languages have also been extensively studied [134]. In particular, a temporal query language on XML-like data is described in [39]. Our subscription language can also be viewed as a temporal query language, although a particular one because of its focus on changes.

Some of the techniques we use originate from active databases [156]. A main distinction is that we are not directly aware of the changes of data on the Web. This modifies somewhat the issues and introduces real challenges in terms of fast reaction to changes [62].

Continuous query system have recently triggered a lot of interest. For instance, The Conquer system [94, 49, 93] can run SQL-like queries on a given HTML document. Like ours, their system provides email notification. However, building a complex SQL-like query with Conquer requires having a solid knowledge of the Web page that is being monitored. The factorization of database triggers is studied in [78]. The optimization of XML continuous queries is studied in [41, 114]. These works are to some extent complementary to ours.

2.3.3 Applications

It should be pointed that change monitoring is becoming very popular on the Web. For instance, NetMind [106] offers change monitoring on HTML Web pages that you submit, and notifies you via email. Some search engines (e.g., Northern Light [116]) also provide such functionalities. Compared to such systems, our main advantage is that because of XML, we can support more elaborate a subscription language with monitoring at the element level. In absence of information, we cannot really compare our performance to theirs on more basic subscriptions.

2.3.4 Updating XML

With the SPIN approach, we store our data in an XML database, Xyleme [162]. The issue of updating XML is a hot topic. Tatarinov et. al. propose in [138] a language and relational database system implementation of XML updates. More work has followed, such as [98] on the more specific topic of version management in XML. For a thorough study on all the XML diff

algorithms, we refer to [48, 46]. In order to equip XML repositories with the same functionalities as relational data management systems, query and update languages have been developed, such as Active XQuery [27], which emulates the trigger definition and execution model of SQL, such as presented in [119]. There are of course many other active database systems, such as Starburst [157] or Hipac [52], but according to [27] SQL3 is the most used in commercial systems.

Our query language is XOQL [11], for more information on XML query languages, and a comparison between five of them, we refer to [28].

2.4 Hyperlink information and semantics

In [95], Simple HTML Ontology Extensions (SHOEs) are described that let web site builders annotate their pages with, indicating what a page represents. We feel that this knowledge is very valuable when searching for information, but it means that each user needs to spend some time describing the pages she builds. Unfortunately, to this day, this task is not performed by Web site managers, and so we feel that this led us to develop the Thesus approach to try to find some concise characterization of a Web page. In the following paragraphs, we detail previous work regarding the potential of exploiting hyperlinks in order to discover pages semantics.

2.4.1 Keyword Extraction from Hyperlinks

The Thesus system essentially works in four stages: i. Retrieval of an initial set of pages from the WWW, which are considered to be on the same subject, ii. Extraction of hyperlink information for these pages from pages than have a hyperlink to these pages (by further querying the WWW based on the initial set of pages), iii. Further enhancement of this information by finding the respective semantics of the extracted keywords then storing this enhanced information locally, iv. Processing of the stored information to discover clusters of Web documents that bear similar semantics and are strongly interconnected. The part of THESUS illustrated in this chapter is mainly related to the areas of link information analysis and querying of the WWW. Additionally since we capitalize on Web documents semantics and clustering based on semantics, we will present similar efforts in these areas.

The issue of extracting keywords from the links is important in the context of this approach and will be further detailed in the following. In [121] the idea of "robust" hyperlinks is introduced. Robust hyperlinks are considered the ones that contain descriptive information on the target document.

This information according to the authors can be limited to five words and empirical results are used to prove this. In [35] some experiments on the hyperlink neighboring text, result in defining the "anchor window" to be 50 characters. These experiments count the occurrence of certain keywords in a window of a given width around the hyperlink. The topic of typing a hyperlink was introduced in [124], but do not adopt an automatic method such as ours.

In [143] a system that increases the Web searching capabilities is proposed, which is based on the idea of attaching information to a document, concerning its concept and its hyperlinks semantics. We propose a structure for hyperlink information with rich semantics. These semantics emanate from a conceptual hierarchy which varies according to the areas of interest and which can be created by domain experts.

Our working hypothesis is that it is easier to characterize a Web page using information provided on pages that point to it instead of using information that is provided by the page itself. As a result, information can refer to the target of a hyperlink, but also to the semantics of the target as they are provided by the source of the hyperlink.

2.4.2 The use of an ontology

In our system, we rely on the use of an ontology and WordNet, in order to compute similarities between words. For more details on ontologies you can refer to [74] and for information on WordNet [159, 13].

2.4.3 Web querying

WebSQL [20] is an SQL-like query language for extracting information from the Web. It enables navigation of Web hypertext either by systematic processing of all the links in a page or through paths that match a pattern, or a combination of both. WebSQL proposes to model the Web as a relational database composed of two (virtual) relations: Document and Anchor. The attributes of Document include: URL, title, full text, size, modification date and type where Anchor includes the base, label and href. Based on an expandable set of Java classes it allows queries over the two virtual relations. Queries serve either information retrieval tasks (i.e. Find the Labels of all Hyperlinks to Postscript Files) or Web maintenance tasks (i.e. Find all broken links in a page, find missing images etc)

Googles advanced search allows the user to include and exclude certain terms that must appear in the title or in the URL of a page. However, results are limited to a certain thematic domain. Google is also able to find

the incoming links of a page or pages similar to the one considered. The ranking algorithm, PageRank [30] is used only to give to most important pages a higher rank, but not to group pages of common interest. Googles PageRank algorithm associates the text of a link with the page that the link points to, thus providing more accurate descriptions of Web pages than the pages themselves, which as a matter of fact, need not to be fetched.

WebWatcher [19] detects the presence of certain keywords inside the scope of hypertext link and the sentence that contains the link and ranks the links according to the users interests. The set of keywords is produced using a training set and is limited to a few hundred terms.

The Automatic Resource list Compilation (ARC) system [35] uses the text around href links to describe the contents of the pointed page and to enhance the hub and authority weight on a certain topic. In particular, if text descriptive of a topic occurs in the text around an href into a page p from a good hub, it reinforces the belief that p is an authority on the topic.

The main assumption is that links to a certain page act as recommendations of this page by authors of other pages. Hopefully, these authors act independently of the page's author and their opinions can be added to the ranking of the page, yet our system could also be able to detect "malicious" links by finding that they have no term in the ontology that sets them close to the ones that appear in other links for instance, and a scheme could be devised to filter them out. According to [81], this kind of ranking is called connectivity-based ranking.

2.4.4 Link analysis and Web document clustering

In most cases Web document clusters are built based on connectivity between documents (Web structure) and not on semantics that the connectivity might convey. On the other hand, Web content mining techniques mainly perform text mining on the whole document while ignoring the structure of HTML documents. It would be very useful to consider both hyperlinks of pages and their contents and then automatically classify large collections of Web documents.

Some works relate to the importance of links as entities that promote semantics in a hypermedia network, such as [91] who show that there is high quality in exploiting the text contained inside the hyperlink area, or [25] that use link information in order to do topic distillation of Web documents.

Kleinberg [83] states " *The link structure of a hypermedia environment can be a rich source of information about the content of the environment (...) But for the problem of searching in hyperlinked environments such as the World Wide Web, it is clear from the prevalent techniques that the information*

inherent in the links has yet to be fully exploited."

There is a consensus that clustering techniques should be applied to the results of a query rather than on the whole Web space, in order to discover groups of relevant documents. In [165] a suffix tree-clustering (STC) approach is proposed where the algorithm based on phrases shared between documents is used to create the clusters. The documents to be clustered are first cleaned (using a stemming algorithm). Then the base clusters are identified using the suffix tree structure. This process can be viewed as the creation of an inverted index of phrases for the document collection. Finally the significantly overlapping base clusters are merged using a similarity measure that is based on the number of documents common to two clusters. The results are encouraging from the efficiency point of view and the algorithm is linear in the number of documents.

Haveliwala et al. [79] propose a clustering approach for Web documents to find groups of similar pages. The similarity is based on words found in the documents along with their occurrence frequencies. The similarity measure adopted represents the ratio of the common terms divided by the union of the terms of two documents. Term matching is exact.

Link information is already used by Web search engines to better filter and rank query results, e.g Google's PageRank algorithm prioritizes pages with many incoming or outgoing links. An interesting server that uses hyperlink's structure to group interconnected results is Kartoo [85]. Vivisimo [145] proposes a clustering approach for Web document organization. It makes use of the contents (titles and brief descriptions) that are returned by the underlying search engines. Northern Light [116] classifies each document within an entire source collection into pre-defined subjects and then, at query time, selects those subjects that best match the search results. Vivisimo does not use pre-defined subjects; its annotations are created spontaneously.

2.4.5 Inferring Web Communities from Link Topology

[68] introduce the notion of hyperlinked communities. Each community contains a core of central pages (authorities) linked together by "hub" pages. The authors show it is possible to derive topic generalization directly from the pattern of the linkage. The authors also claim that the high-level structure of the Web is in really very structured, or in any case, is much easier to analyze than local level.

Flake[66] defines a community as a set of sites that have more links in either direction to members of the community than to non members, and use a max flow / min cut framework in order to find the most important members of the community.

2.5 Clustering

For a general overview on clustering, which is far beyond the scope of this thesis, we refer to [96, 16, 126, 135, 158]. In this section, we will specifically detail two important aspects : related work on the similarity measure that we use, and previous work on the specific topic of the clustering of Web documents.

2.5.1 Similarity measures

Existing similarity measures between sets A number of similarities/distances between sets of elements already exist in the literature [60, 73]. The most widely used one, the Jaccard coefficient is simple: Let A and B represent two sets of elements. The similarity between A and B is defined as:

$$S_1(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

In [60] the problem of measuring the similarity or distance between two finite sets of points in a metric space is considered. Some of the distance functions are reviewed, amongst them the minimum distance link measure, the surjection measure, and the fair surjection measure. All of them take polynomial time algorithms for their computation. In order to be competitive, our measure should also try to be tractable in polynomial time.

In [73] the idea of calculating similarities between sets, using the Jaccard coefficient, is investigated. The indexing issue for distance / similarity between sets of values is treated in recent work [69], again using the Jaccard coefficient to calculate the similarity. [26] also investigate this with their mediator approach.

The traditional cosine measure from the Information Retrieval literature (see [131]) has the same behavior as the Jaccard coefficient. As a matter of fact, it can be viewed as a direct application of the Jaccard coefficient.

However, in all the aforementioned efforts, the sets of values are considered as flat (i.e. all values are independent from each other) therefore only exact matching between values is taken into account by the Jaccard coefficient. With our approach - defining a document by terms of the ontology, and calculating the similarity between these sets of terms – documents defined by different sets of words may end up having a very high similarity rating. For instance, recall the example given in the introduction, two documents defined by (cat, food) and (feline, diet) would have a similarity value different from 0, which is its similarity value according to the Jaccard coefficient. In

our case, we want to take into account the proximity that can be derived by the distance between terms in a tree - in our case an ontology.

Similarity between two elements of an ontology

In order to compute the distance between sets of terms of an ontology, we were lead to investigate the existing similarity measures in the more simple case of calculating the similarity between two given terms of the ontology. [129] and [127, 128] propose different measure inside a taxonomy such as WordNet, and [92] proposes a comparison between these measures and others, such as Wu and Palmer [160], Miller and Charles, and a novel similarity measure. [54] also propose the use of Wu and Palmer measure in the ontology context. The Wu and Palmer measure is the fastest to compute, and is arguably as good as the others (see [92]), which is why we chose to use it. We detail its definition and properties in the following paragraph.

Wu and Palmer similarity measure: Given a tree, and two nodes a, b of this tree, we first of all find their deepest (in terms of depth in the tree) common ancestor c. The similarity measure is computed as follows

$$S_{WP}(a,b) = \frac{2 \times Depth(c)}{Depth(a) + Depth(b)}$$

2.5.2 Clustering Web Documents

Clustering Web documents solely on their text content is not very effective when trying to construct distinct groups. Work closely related to this thesis is presented in [80]. He, Zha, Ding and Simon use Hyperlink Structure, cocitations and textual content in order to construct a novel similarity metric for measuring the homogeneity of documents. They then use a normalized-cut, which the authors show is in fact a variation of the K-means clustering method in order to cluster Web documents. Although He *et.al.* take into account link structure, they do not, like we do, exploit the semantics contained in the text close to the hyperlink area. Furthermore, their similarity measure is not documented.

Another relevant project is [122]. Using both topological and textual similarity between pages in order to form document collections, the authors classify nodes in order to construct functional categories or types[5]. This approach is however restricted to a given site, and a given topic. We are not aware of any larger scale experiments using these algorithms.

[5] A category might be an index page of a web site which is used mainly for navigation, or a reference page which is a page that explains a concept, etc...

K-means clustering algorithm We detail the clustering algorithms used in our system, DB-Scan[61] and CobWeb[65] in Chapter5, but let us briefly here give explain the general way K-Means works:

1. Partition all the elements to be clustered into N clusters, where N is the final number of clusters to be obtained.

2. For each element, calculate the cluster it is closest to, and reassign it to that cluster

3. Repeat step 2 until convergence

In order to run K-Means, the algorithm needs to have a way of calculating which cluster a given element is closest to. The canonical way of doing this is to consider the number of closest neighbors in each cluster a document has, and to assign it to the cluster that has the greatest number of close neighbors, assuming of course, that a similarity measure between two elements has been defined.

Let us stress that whereas K-Means iterates over the whole dataset until convergence in the clusters is reached, CobWeb works incrementally, updating the clustering instance by instance.

Chapter 3

The Architecture of a Web Warehouse

Related Publications: Thesus architecture: [144, 111, 113]; Xyleme Architecture: [163]; Spin: [5, 1]

3.1 Introduction

In this chapter, we present and analyze two large scale Web warehousing applications, that have been implemented, and we propose a generic service oriented model to construct modular Web warehouses. We first present in Section 3.2 the architecture of the Thesus system. This system is a warehouse of Web data on a specific topic. I developed it in collaboration with M. Vazirgiannis, I. Varlamis and M. Halkidi of the Athens University of Economics and Business, participating in all stages of the design and implementation. We will detail in this chapter its modularity, and some of the functions in this chapter. In Section 3.3, we show the architecture of the Xyleme system, that was developed at INRIA, in collaboration with partners such as University of Mannheim, Conservatoire National des Arts et Métiers, Paris, and Laboratoire de Recherche en Informatique, of the University of Paris XI. In the Xyleme project, I was specifically in charge of the monitoring module, which is the topic of Chapter 4.

In Section 3.4, we detail a generic service oriented approach to construct Web warehouses, using pre-existing modules. This approach is based on the observation that all Web warehouses have many functionalities in common, and may share a modular architecture.

3.2 The Thesus Experiment

A complete description of the Thesus system is beyond the scope of my thesis.

In this section, we present the architecture of the Thesus system. The system components are detailed in this section. Specific details and experimentation regarding the document enhancement module, which attracted the most of my work is detailed in 5.5.2.

The main objective of the Thesus system is the characterization of a set of Web documents using hyperlink information, enhancing its characterization with semantics and distilling the set into thematic subsets (Thesus). The objective is to define a Thesu, putting emphasis on the link semantics and the connectivity features among the collection of documents. Once processing is completed, we are can submit queries on the Thesu that can return hubs, authorities, and co-citations enriched with the semantics of the links. The developed clustering algorithms further divide the Thesu into smaller subsets of pages with similar semantics, thus refining the intention of the initial Thesu.

A prototype system has been implemented, and a demonstration has been given in [112]. The system's components include (see Figure 3.1):

- The "information acquisition" module gathers a set of URLs that seem to be relevant to the topic under consideration. This happens by expanding an initial set D of pages generated by a traditional search, or by using a given set of on-topic documents. Expansion is performed as multiple iterations of the "root set generation" algorithm of HITS [83], limited to those links that display on-topic keywords in an expanded window of hypertext. In opposition to the full focused crawling [34], this crawling is totally unsupervised and is not based on exact matching between found keywords and ontology terms. After a maximum of N repetitions, where N is specified by the warehouse designer, an extended set of documents E is created. E contains documents that fall into the same thematic area.

- The "information extraction" module, for each hyperlink of a page, extracts keywords within an expanded hypertext area around the hyperlink. Parameters such as the size of the text area to be analyzed, or the stop words to be removed are ajustable.

- The "information enhancement" module enhances extracted hyperlink information with semantic information by mapping extracted keyword sets to concepts in an ontology, that has been provided by the user.

- The "clustering module" partitions the set E into semantically coherent subsets based either on the extracted keywords or on the concepts they represent. Similarity is computed using a novel measure that employs similarity of sets of terms in a hierarchy and not exact matching of terms. This module is the topic of Section 5.5.2.

- The "query engine" that enables searching in the set E taking advantage of the clusters that are closer to the user's query and ranks the results accordingly, it also supports link analysis operators, thus allowing the search for thematic authorities, hubs, cocitations and couplings.

These modules access a relational database in which the document information is stored and employ knowledge from the WordNet 1.6 database [159] and the ontology. The system has been evaluated on different experimental Web document sets of the order of 10^4 documents, a reasonable size for a personal thematic warehouse.

The system provides the following:

- The creation of URL collections that are characterized by certain keywords. This is achieved by combining simple Web searching services provided by search engines (i.e. search for pages that contain specific keywords) and a thematic crawler that expands the search engines results.

- The characterization of Web documents or Web subsets either at keyword or at semantics level through the processing of their incoming or outgoing links (by processing links neighboring text in the source URL)

- The clustering of characterized sets of URLs into subsets based both on their characterization or on their intra-connectivity graph.

Not surprisingly, the acquisition and processing of a large corpus of Web documents is a rather lengthy task. Therefore the first three modules of the system are mainly used off-line. The query module operates as an end-user tool that allows browsing and search the enriched information repository. See section **??** for details on running times for the various tasks.

A Web service is also available via www.db-net.aueb.gr/thesus that demonstrates the hyperlink information extraction module capabilities and the operation of the three different implementations of groupSemantics operator. A Web interface is also available that allows users to access a sample database on Music created by Thesus. Users can test the operators of the Thesus language to query the enhanced information.

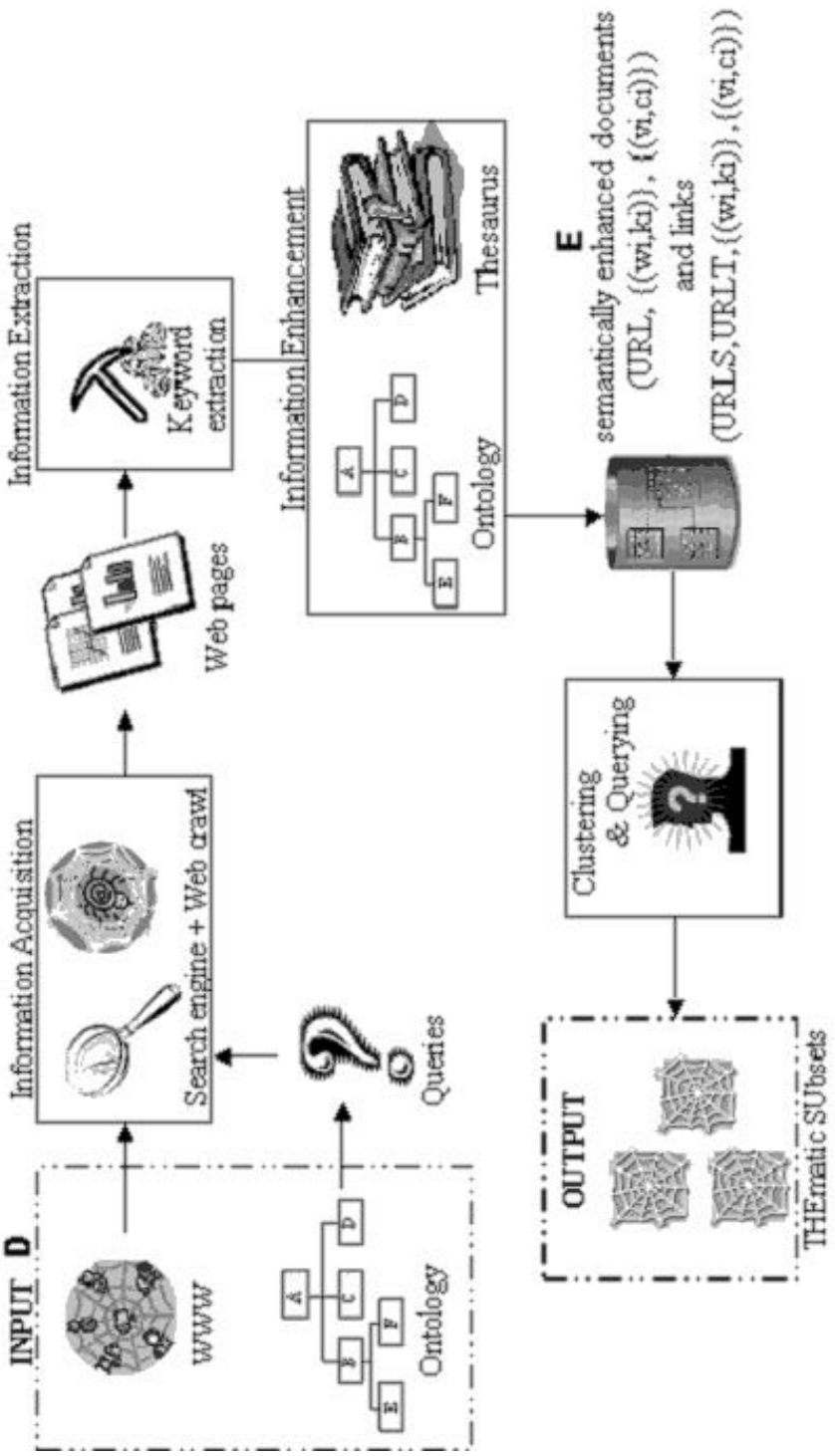


Figure 3.1: Thesus Functional Architecture

Technical note: The crawling system was developed as a multi-threaded Java application. Information extracted from Web pages and related semantics are stored in a relational DBMS (Microsoft-SQL Server). The access to the database is performed using a JDBC native driver. The client application remotely connects to the database server using JDBC and allows users to perform the Thesu language operations on the collected data.

We next detail the main functionalities of Thesus.

Information acquisition

We assume that an ontology O over a certain thematic domain (i.e., Arts, Music, Technology, etc) is given. The thematic crawler generates the core-set of URLs in two ways, depending on the existence of an initial set of documents D of interest.

If D is given, the crawling starts from this set without using any Web search engine services. The crawler visits all documents (URLs) in D, extracts information from their outgoing hyperlinks and keeps only hyperlinks that are characterized by at least one concept in the ontology. We say that a hyperlink is characterized by a concept when the anchor-text area around and inside the hyperlink contains a keyword that is mapped to the specific concept in the ontology (the operator getSemantics implements this, the algorithm used takes into account not only proximity between a word and a concept, but also the proximity of groups of words to one concept, see 5.5).

The targets of such hyperlinks are suggested for the next crawling step. This process stops after a certain depth of crawling is reached or after all documents suggested for crawling in a step have already been considered. Due to the weak connectivity of the World Wide Web documents (only 28% of the Web is very strongly connected according to [31]) it is possible that with a small ontology (i.e. less than 20 concepts) and a small initial set, the crawling process will "naturally" stop. Most of the time, it will be interrupted at a certain depth, a parameter specified by the warehouse designer.

If D is not given, we consider no prior knowledge on the specific domain, so we use a Web search engine to locate some documents that are used as seeds for the process. In this case, the only input is the ontology O that describes the users interests. As per definition **??**, the ontology considers only the generalization-specialization (IS-A) relation between concepts, thus forming a hierarchy of concepts starting from the more generic and branching to the more specific. The system generates the paths from the root to all the nodes in the ontology. Then it queries a Web search engine for all the generated paths (node1 AND node2 AND AND nodeK) and collects the top T answers returned for each query by the search engine. In our experimenta-

tion, we set $T = 50$. If the ontology contains N concepts, then N paths and respectively N queries are generated. If the search engine returns a maximum of M answers the total number of URLs in D will be fewer than $M \times N$. Experimentation shows that the results are usually highly overlapped so the final number of documents in D after removing duplicates is significantly less than $M \times N$.

Information extraction

Our first task to associate some basic semantics (keywords) to pages is based on the links pointing to them. In order to enrich the information on the core collection of URLs, we crawl one level backwards and collect the incoming links information, i.e., links pointing to the documents in the collection. The crawl({URL},-1) operator requires the use of a Web service that gives the top N incoming links of a page. The documents are processed as follows to enhance their classification quality. The first step is to process the incoming links to D and extract keywords from the respective source pages. Calling the `weightedGroupKeywords` operator for the documents $\{b\}$ that point to each document d_i in the set we get a set of pairs of keyword / times of occurrences $\{(k_{d_i}, t_{k_{d_i}})\}$ that characterize d_i. This operator is detailed in Section 5.4.4.

$$For\ every\ d_i \in D\{(k_di, t_{k_di})\} : groupSemantics(\{b\}, \{di\})$$

In later sections we experimentally verify the potential of this approach.

The issue of extracting keywords from the links is important in our context. Taking into account the works of [121, 35], in Thesus, keywords are extracted from the link anchor (i.e. the string between the $< a >$ and $< /a >$ tags) and from two text strings (each of 100 characters long), one preceding the starting link tag and another following the ending link tag. The window is trimmed whenever certain HTML tags appear, such as $< a >, < li >, < tr >, < td >$ etc, because such tags usually signify the logical end of the hyperlink neighboring area. The window size has been chosen *after* experimentation so that the resulting mean number of keywords is approximately five [121]. The algorithm is the following:

1. Find the limits of the hyperlink area (start, end position)
2. Find the limits of the pre-hyperlink area (start-100, start)
3. Find the limits of the post-hyperlink area (end, end+100)
4. Search for image tags in the inside hyperlink area.

   (a) Extract keywords from the *alt* attribute of image tags

   (b) Remove image tags

   (c) Remove punctuation, stop-words, small words

   (d) Extract keywords from the remaining text

5. Search for special HTML tags (table row, table column, hyperlink, list item etc) in the pre-hyperlink or post-hyperlink area. Since such tags denote a change on the document structure, they should limit the hyperlinks neighboring area.

   (a) Remove complete HTML tags inside the area. For example $< B >$, $< I >$, $< IMG >$ etc.

   (b) Remove punctuation, stop-words, small words

   (c) Extract keywords from the remaining text

Let us note that other information, such as the title of the Web page, or the presence of some specific words in the URL could also be used in order to extract semantics that characterize the page. This has been proven to work well, and is used in some algorithms such as Google's. In our case, we prove that the simple use of incoming links is gives good results, but using extra information extraction techniques is of course possible.

Let us give an example of how the algorithm works: In the HTML fragment that follows, keywords are extracted from the hyperlink to the CERN home-page:

```
<A HREF="http://cts-hp.cts.iisc.ernet.in/"> Centre for
 Theoretical Studies</A><D>Indian Institute of Science,
 Bangalore, India<DT></LI>
<LI><A HREF="http://www.cern.ch/">CERN</A><DD> European
Organization for Nuclear Research/European Laboratory
for Particle Physics, Geneva, Switzerland (See also <A
```

From the hyperlink to the URL: http://www.cern.ch the phrases extracted are:

- The hyperlink area that gives the phrase "CERN"

- 100 characters preceding the hyperlink. The area is trimmed where $< LI >$ appears so no characters remain as a matter of fact.

- The 100 characters following the hyperlink. The last partially selected keyword (Swi) is ignored, $< DD >$ is removed, stop word "for" is removed also. The remaining phrase is: "European Organization Nuclear Research/European Laboratory Particle Physics, Geneva".

So the set of keywords that are extracted describe this specific target URL with the number of appearances is {(cern,1), (european,2), (organization,1), (nuclear,1), (research,1), (laboratory,1), (particle,1), (physics,1), (geneva,1)}.

Enhancement, clustering and Query modules

We will not detail in this Section the specificities of these three modules, since they are explained in greater detail in Chapter 5.

Conclusion

We have described here the architecture and main services of Thesus. The system is modular, each module can work separately, if provided with the required input. The "information acquisition" module, which implements the `semanticCrawl(URL,keyword,N)` operator (see Definition 5.4.6) can alternatively use a set of ontology concepts as the input keyword set and an initial set of pages (preferably with many outgoing links) and give a collection of thematicly close URLs as output. The "information extraction" module takes as input a set of URLs and characterizes it by extracting keywords from the links pointing to or from this set with the use of the advanced link semantic operators. The "information enhancement" module takes as input a set of documents characterized by sets of keywords (weighted or not) and an ontology and produces a semantic characterization of the documents by mapping the sets of keywords to sets of ontology concepts. Finally the "clustering module" clusters a set of characterized documents (either with keywords or with semantics) to subsets containing similar documents, using WordNet and the ontology.

3.3 Xyleme

A complete description of the Xyleme system is beyond the scope of my thesis. We only give here a brief and partial description of the main functionalities of the system.

The main modules are shown in Figure 3.2. The lowest layer consists of the XML *repository* and *index manager*. The first version of the repository, called Natix, was developed at Mannheim University. It is tailored for storing

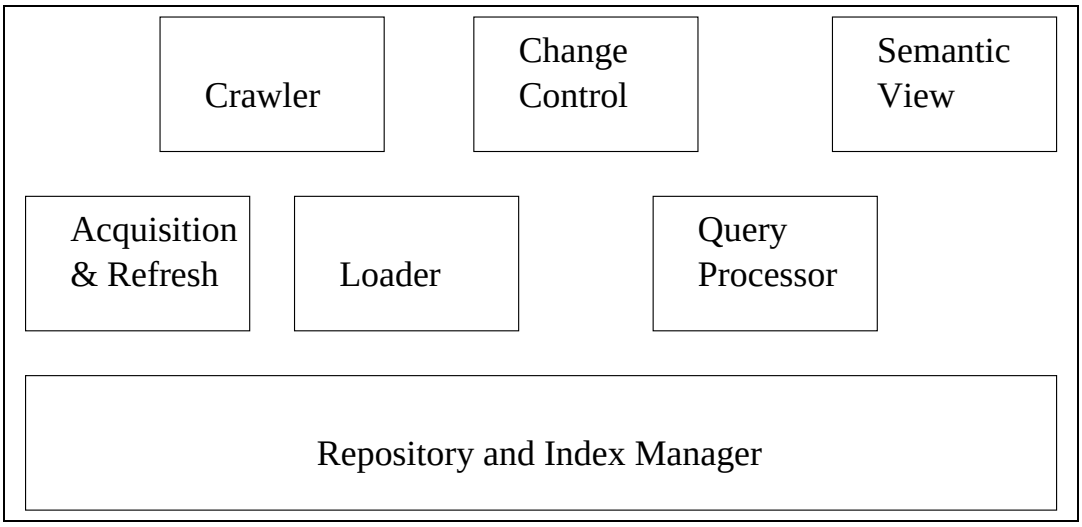


Figure 3.2: Xyleme Functional Architecture

tree-data [107], e.g., XML pages. Above the repository, on the left hand side, are the three modules in charge of populating the XML warehouse. Since the functionalities of both *crawler* and *loader* should be obvious, let us just say a few words about the *data acquisition and refresh* module [98, 103]. Its task is to decide when to read an XML or HTML document, and possibly when to refresh it. This decision is based on criteria such as the importance of a document, its estimated change rate or subscriptions involving this particular document. On the right-hand side, we find the *query processor* [12] that is an XML-tailored query processor that supports an XML query language. Above it are the *change control* and *semantic* modules. The main role of the *semantic* module is to classify all the XML resources into semantic domains and provide an integrated view of each domain based on a single *abstract* DTD for this domain. The operations performed by this module are detailed in [53]. It is important to note that while the Thesus module enriches a Web document, by characterizing it with terms of an ontology, the Xyleme Semantic module builds upon the fact that XML documents of a given domain are going to have similar tags, or tags with the same semantics, and can be, as such grouped and queried together.

Xyleme runs on a cluster of Linux PCs [120]. All modules and in particular the XML loaders and the indexers are distributed between several machines. The repository itself is distributed. Data distribution is based on an automatic semantic classification of all DTDs. The system tries to cluster as many documents as possible from the same domain on a single machine.

The entire system is written in C++ and uses Corba [50] for internal communication and HTTP for external ones.

The following paragraphs detail the various modules of the architecture. I was specifically in charge of the *change control* module, that will be the topic of Chapter 4.

Repository: Xyleme requires the use of an efficient, updatable storage of XML data. To this end, the system uses Natix, developed at the U. of Mannheim. Typically, two approaches have been used for storing such data: (i) store XML pages as byte streams or (ii) store the data/DOM tree in a conventional DBMS. The first presents the disadvantages that it privileges a certain granularity for data access and requires parsing to obtain the structure. The second artificially creates lots of tuples/objects for even medium-size documents and this together with the effect of poor clustering and the additional layers of a DBMS tend to slow down processing. Natix uses a hybrid approach. Data is stored as trees until a certain depth where byte streams are used. Furthermore, some minimum structure is used in the "flat" part to facilitate access with flexible levels of granularity, Also, Natix offers very flexible means of choosing the border between the object and flat worlds. Natix is built, in a standard manner, on top of a record manager with fixed-size pages and segments (sequences of pages). Standard functionalities for record management (e.g., buffering) or fast access (e.g., indexes) are provided by Natix. One interesting aspect of the system is the splitting of XML trees into pages. For specific information on this topic, we refer to [107].

Query Processing: The evaluation of structured queries over loosely structured data such as XML has been extensively studied during the last ten years. The techniques differ slightly depending on the system that is used to store the data, object-oriented, dedicated or relational. However, none of the existing approaches scale to the Web. Xyleme is considering billions of documents (Google is indexing more than one billion HTML documents) and millions of queries per day. So, query processing acquires in Xyleme a truly distinct flavor. As is often the case, the queries are represented using an algebra. The novelty of the approach lies in (i) a distribution pattern that scales to the Web (ii) the efficient implementation of a complex algebraic operator, named PatternScan, that captures so-called tree queries, i.e., queries that filter collections of documents according to some tree pattern and extract information from the selected documents. The added expressive power provided by database-like XML query languages is captured using standard

operators (e.g., Join, Map, etc.). For details on this topic, we refer to [12].

Data Acquisition: Xyleme needs to crawl the Web in search of XML data. It also needs to refresh pages to keep the repository up to date. To this end, a crawler that can read millions of pages per day was designed. Several crawlers can be used simultaneously to share the acquisition work. Note that only XML pages are stored. For HTML, they are read in order to discover new links. A critical issue is the strategy to decide which document to read/refresh next. First, a Xyleme administrator is able to specify some general acquisition policies, e.g., whether it is more urgent to explore new branches of the Web or to refresh the documents we already have. The system then cycles around all pages it knows of (already read or not) deciding for each page whether this page has to be refreshed/read during this particular cycle. The decision for each page is based on the minimization of a global cost function under some constraint. The constraint is the average number of pages that Xyleme is willing to read per time period. The cost function is defined as the dissatisfaction of users being presented stale data. More precisely, it is based on criteria such as:

- Subscription and Publication: A user may specify a desired refresh frequency for some particular documents. Also, some query subscription may request the regular refreshing of some pages. Finally, the owner of a document may let Xyleme know when a change occurs and request a refresh of the warehouse for this document.

- Temporal information such as last-time-read or change rate: This is an estimate of the number of updates that were missed for a particular page.

- Page importance: The system must read in priority documents that are "important" and refresh them more often than others. To estimate page importance, the structure of the Web and the intuition that important pages carry their importance to pages they point to are used. This leads to a fix point computation, which is detailed in [7]. More details on the crawler itself can be found in [123].

Change Control: The first aspect of change control we considered is a subscription mechanism [109]. At the time we load or refresh a document, we detect whether it verifies patterns specified by some subscriptions. Examples of changes that can be monitored include, for instance, a modification of a particular document, the discovery of new documents with a given DTD. We

also support monitoring conditions at the element level, e.g., monitoring of the newly discovered documents with an element tagged address containing the word "Marseilles". We will show further, that the monitoring system we developed allows to monitor, on a single PC, the loading of millions of documents per day, with millions of subscriptions.

The second aspect of change control that was in Xyleme is a versioning mechanism, developed by Gregory Cobena and Amelie Marian, and presented in [48]. For each page, Xyleme gets snapshots of the page. It is possible to version some pages. To represent versions, a change-centric approach was used, as opposed to a data-centric one that is more natural for database versioning, more precisely, deltas in the style of Hull's deltas [57]. Given two consecutive versions of a page, the delta is computed using a very efficient diff algorithm. As often done in physical logs, completed deltas that also contain additional information so the deltas can be reversed and composed are also stored. The monitoring can also work on changes over these deltas, but we will not detail this.

Semantic Integration: Queries are more precise in Xyleme because, as opposed to keywords searches, such as in Thesus, they are formulated using the structure of the documents. In some areas, people are defining standard documents types or DTDs, but most companies publishing in XML often have their own. Thus, a modest question may involve hundreds of different types. Since, it cannot expected that users know them all, Xyleme provides a view mechanism, that enables the user to query a single structure, that summarizes a domain of interest, instead of many heterogeneous schemata. Views can be defined by some database administrator. However, this would be a tedious and never ending task. Also, meta-data languages such as RDF may be used by the designer of the DTD or by domain experts to provide extra knowledge of a particular DTD. It is probable that this will become common in the long run. The RDF Suite [15], for instance, which is a registry of RDF schemas is one of the first steps in this direction. Thus, the semantic integration team studied how to automatically extract it using natural language and machine learning techniques. The results of their work can be found in [53].

3.4 A modular and service oriented approach : The Spin (Sets of Pages of INterest) project

Both systems we have briefly presented have a certain number of features in common, both from the architecture point of view, and the component point of view. Based on our experience with Xyleme and Thesus, we decided to find a way of building a generic, modular Web Warehousing application. The core idea is the following : we will use a semi-structured model to represent the data stored in the warehouse, and the all the stages of the processing will be conductedusefulfuhf by Web Services.

We will detail the following topics:

- *A declarative specification:* We propose a language to easily declaratively specify a dynamic warehouse of Web resources. The language describes on the one hand the logical organization of data and metadata. The language also specifies the services and queries that manage it. The model relies on XML types to describe pieces of information.

- *A system:* We present the Spin system that can be used to construct and maintain a warehouse based on its declarative specification and user feedback. The system consists in a *Warehouse-compiler* and an execution platform. The platform is based on standard tools (e.g. Java runtime) and on more specific tools, such as the ActiveXML [2] engine. ActiveXML is a system that enables the management of XML documents with embedded Web service calls (in the spirit of Sun's JSP or Microsoft ASP). The compiler generates an instance of the warehouse (some XML schemas and some initial XML content) and the code that will manage the data of the warehouse and the calls to the Web Services.

  We use an ActiveXML execution platform. We also provide some high level requirements that should be supported by any platform to run the *Spin* system (e.g., what the platform needs in terms of storage, querying and update functionalities). For instance, one could develop such a platform simply in Java.

- *A simple way of querying results:* An application server based on an XML query language is used to browse the warehouse, select and process content. Users may also provide feedback to the warehouse engine by updating some of the warehouse data. This feedback may be used by other services.

- *Temporal Evolution:* The warehouse should reflect the actual state of the changing Web. In this spirit, the system supports change control (versions, query subscription) that grant evolution capabilities to the warehouse.

- *A library of warehouse services:* We provide a set of services that are frequently used by warehouse designers such as Web-crawling, classification, page ranking, version Management, etc.. We also mention some that are more specific to a particular project, namely *e.dot*. The *e.dot* project is sponsored by the RNTL program of the French Government [59].

In particular, in Section 3.4.1, we present the motivations for this work. In Section 3.4.2, we describe the architecture of the system. In Section 3.4.3, we introduce the specification language. In Section 3.4.4, we discuss the prototype.

3.4.1 Motivations

Context The context of our work is the *e.dot* project : the construction of a warehouse on the topic of *food-risk assessment*. More precisely, one goal is to create a warehouse consisting of data to be used by biology researchers. Some of the data is already structured and comes from relational databases, managed by partners in the project. The rest of the data has to be discovered and retrieved from the Web, and then stored for further analysis. Because of the focus on Web resources, we call such a warehouse, a *Web-Warehouse*. To build it, we use the Spin system that provides the following functionalities:

- retrieval of data of interest, in particular from the Web.

- analysis of its content, (i.e., extraction of numerical values of bio-experiments described in the Web documents).

- validation of Web data by technicians by providing them with context information: the data source (e.g. Web site), the topic or category of the documents, and so forth.

- integration of Web data and of relational data.

Figure 3.3 shows a warehouse that is constructed using the system. It stores both Web data and relational data. Several Web services are used to clean, filter, and enrich the Web data. Users (bio-technicians) are also involved in the dynamics of the warehouse since they (in)validate some of

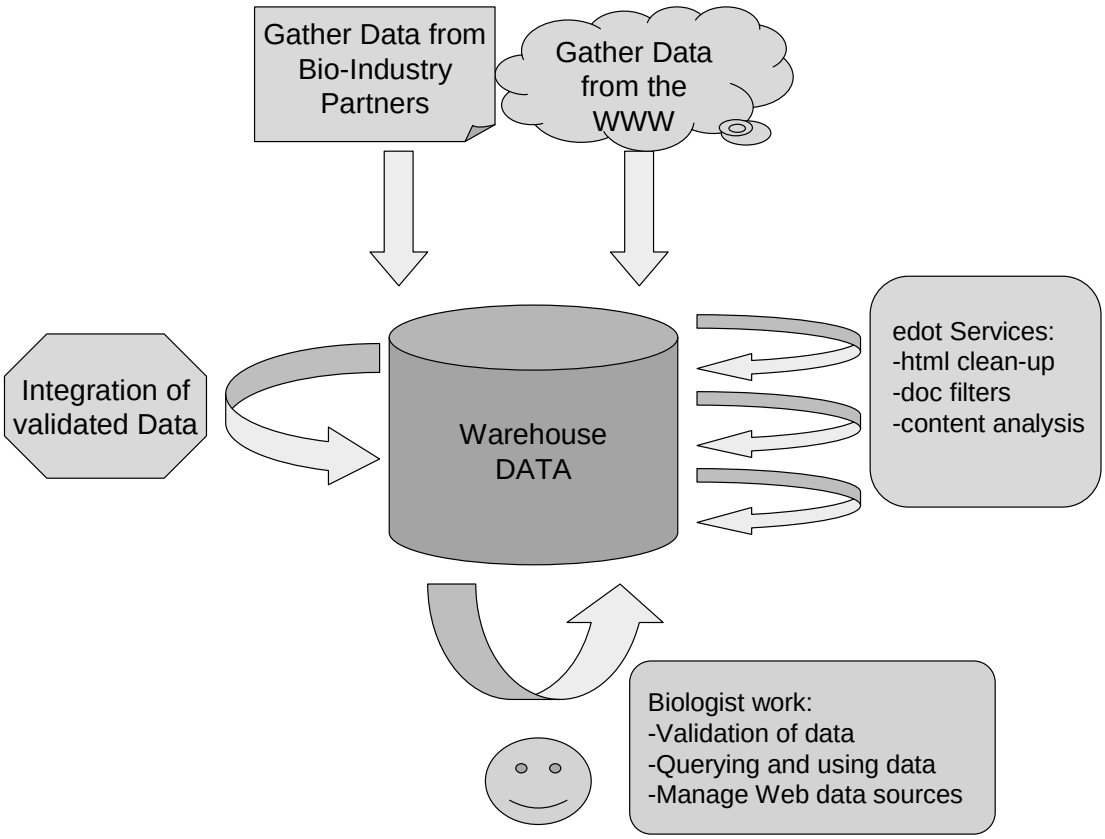


Figure 3.3: *e.dot Warehouse*

the Web data. For instance, they may approve a well known reliable source of data (*http://www.pasteur.fr/*). They may also annotate specific keywords (e.g. a list of bacteria) that are used by the crawler to query search engines. Automatic tools are also used to validate the Web data against the relational DB, such as automatically extracting data from more structured parts of web pages, such as HTML table, or Excel spreadsheets. Finally, we provide querying tools to exploit the data.

3.4.2 The Architecture of the Spin system

In this section, we briefly present the architecture of the system.

The system is built to support high-level warehouse specifications in a dynamic language that will be considered in the next section. The language is dynamic in the sense that it enables the use of queries and Web service calls to build, and update the content of the warehouse. It is also possible for a user to interact with the information stored in the warehouse, e.g., by

annotating pieces of information. The data and meta-data in the warehouse are stored in XML format.

As was just mentioned, a number of technologies are needed to construct and maintain such a warehouse. The approach we follow is based on: (i) XML to store and exchange data, (ii) an extensive use of Web services [147, 148], (iii) XML query languages (XQuery [150] or XOQL [11]) and update languages (XUpdate [161]).

The Spin system consists of two main components:

- The compiler that is used to compile a high-level specification into ActiveXML documents. ActiveXML [2] is an extension of XML with embedded services calls. Since ActiveXML is built around XML, the ActiveXML documents have a structure that is reasonably close to that in the logical model. Service calls are introduced in the ActiveXML documents to take into account the processing required by the application.

- The running platform is simply an ActiveXML server. The system is completed with the appropriate Web services described in WSDL [148] that are used in this particular warehouse. These services may either be part of the system or be accessed remotely.

An overview of the architecture is presented in Figure 3.4.

3.4.3 The warehouse model

In this section, we present the main aspects of the language that is used to specify a warehouse. This language has two distinct facets: one for data modeling and one for service modeling. The language is a semi-structured (XML) DTD, we will give some examples in the following sections. The DTD itself can be found in Appendix **??**.

Data modeling

We next illustrate the model with an example. We present here a possible definition of a simplified warehouse for Web documents.

Example We first define the simplified Data Types for textcontent (the content of some Web page), author, authors, URL, document are defined using "basic" data types such as `string` or `integer`:

```
<dataTypes:define datatype="textcontent">
  <dataTypes:child type="string"
```

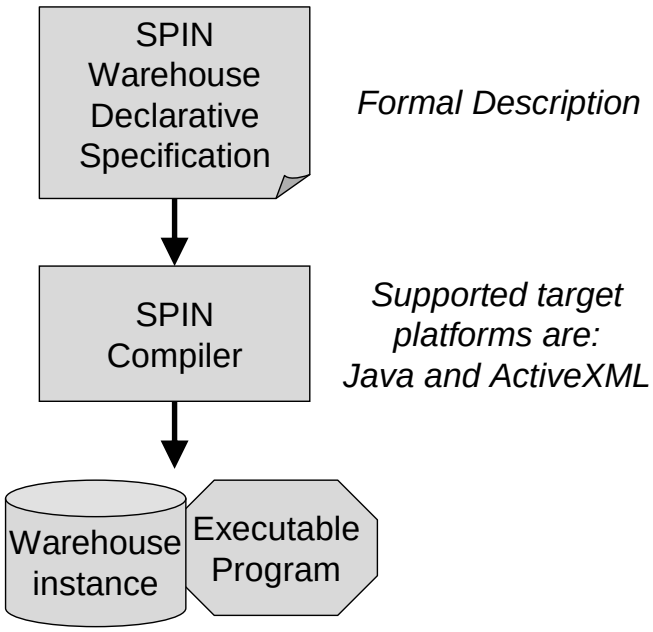


Figure 3.4: *Spin Architecture*

```
      name="value"/>
</dataTypes:define>
<dataTypes:define datatype="URL">
  <dataTypes:child type="string"
      name="value"/>
</dataTypes:define>
<dataTypes:define datatype="document">
  <dataTypes:child type="URL" name="URL"/>
        <dataTypes:child type="textcontent"
            name="content"/>
</dataTypes:define>

<dataTypes:define datatype="author">
  <dataTypes:child type="string"/>
</dataTypes:define>
<dataTypes:define datatype="authors">
  <dataTypes:child type="collection"
       of="author" key="author.value"/>
</dataTypes:define>
```

Then, we define the warehouse. The warehouse is a *collection* of *documents*. We define here the key for documents : `document.URL.value`. Each document is enriched by entities such as *authors*, *experiment*, *measure*, *bac-*

teria. The general organization of this warehouse may be given by: .

```
<Whouse:collection name="TheWarehouse"
            key="document.URL.value">
  <Whouse:entity datatype="document" >
    <Whouse:entity datatype="authors" />
    <Whouse:entity datatype="experiment">
      <Whouse:entity datatype="bacteria"/>
      <Whouse:collection name="measures"
                  key="measure.item">
        <Whouse:entity datatype="measure" />
    </Whouse:collection>
    </Whouse:entity>
  </Whouse:entity>
</Whouse:collection>
```

The data facet of the model first supports the definition of *DataTypes*, i.e., the structure of XML data fragments, that will represent pieces of information. The warehouse then organizes information in a tree-hierarchy (in the style of a directory hierarchy) of entities where each entity is an instance of a DataType or a collection of such entities. The relation between an entity and its children is *enriched by*. The relation between a collection and the unique type of its components is *collection of*.

An important aspect in the model is that each piece of information in the warehouse can be uniquely identified. This is in particular based on a careful naming of DataTypes and entities (details omitted). The other aspect is that for each collection, a key (for its components) is specified that identifies each of the items of the collection. This is in the spirit of [33]. In a nutshell, a key is an XPath expression, relative to each element in a given collection that locates the XML element that will be used to differentiate each element in the collection. Smart naming of entities and collections grants us the ability to write meaningful XPath [149] expressions that identify each node.

Note that it is possible to omit the name of some entities that are then taken by default as the name of the corresponding DataType.

Starting from such a specification, the compiler does three things:

1. it merges the type model and warehouse description into a single description (in XML).

2. it generates a corresponding schema.

3. it transforms abstract queries in the specification of processing tasks (see further) into (XPath) queries that run on the warehouse schema.

Note that there is no fundamental separation between the DataTypes and the hierarchy of collections and entities. The same piece of information could be described in one or the other. However, we accepted the redundancy introduced by DataTypes because they simplify the reuse of software components in different warehouses. Intuitively, the warehouse model consists of the conceptual model of the warehouse, whereas the type model represents the software-level implementation design.

Service modeling

Web services are used to perform processing for the warehouse possibly on remote machines. A service takes some XML input parameters, and returns some XML output. In order to properly use a Web Service, there are three important questions to answer: *when* to call the service, *what* is the input, and *where* should the output be placed. We provide a GUI for the designer of the warehouse that is in charge of specifying this.

In our setting, the input parameter is defined as an XML query over the warehouse data. This query is specified by the designer of the warehouse who knows how the Web services should be used. One important role of the specification GUI is to simplify the use of particular Web services so that the designer does not have to explicitly construct the query. More precisely, the GUI automatically generates the skeleton of a query that the warehouse designer has to fill.

It is the system that is in charge of calling the Web service when needed, e.g. when the input data (result of the query) has changed. For performance reasons, the platform running the compiled *Spin* system should provide mechanisms to control Web service activation. We participated in improving ActiveXML in such aspects. If the query returns a list of results, the service is called several times to process each result in turn.

It is also the role of the warehouse designer to specify where output results should be placed. To do so, for each Web service, the designer must specify a list of update queries that describe what to do with the returned data. More precisely, we apply an update script that is described in the XUpdate (XML Update) language. This update script is itself based on the evaluation of a query (XQuery) over the service output. Note that when XQuery will fully support updates, it will be possible to use directly XQuery (with update facilities) instead of the XUpdate script.

It is often the case that the output of some service call should be placed close to the input. For instance, if a service call finds the list of bacteria in some document, we want to place that list as a child of the document URL node. This can be done as follows. The query that defines the input of the

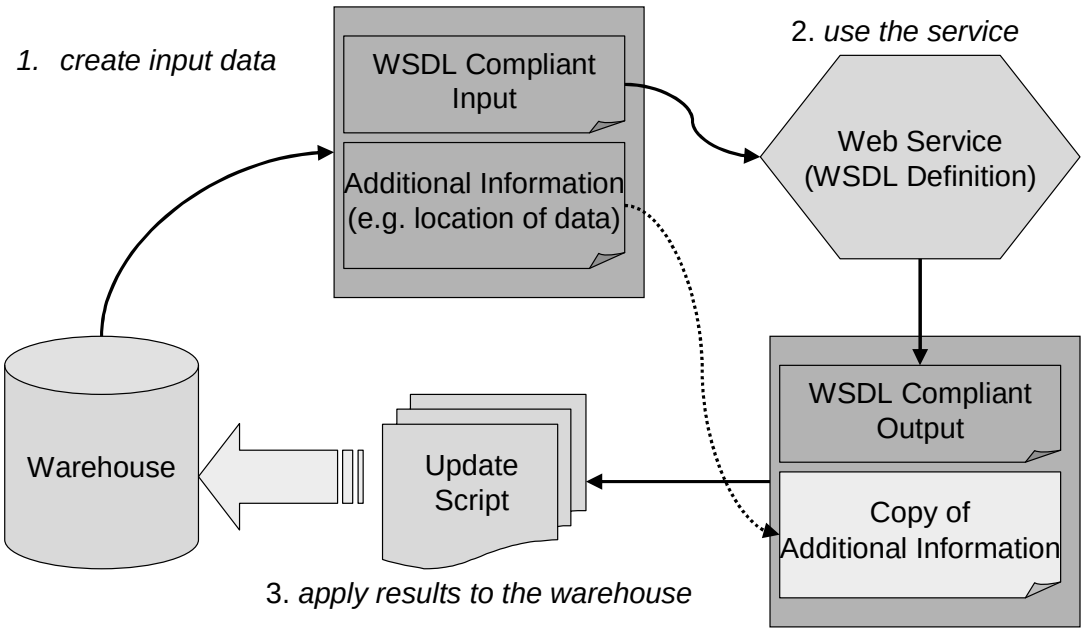


Figure 3.5: *Service Model*

Web Service is also used to generate information about the location of the input data. This location information is provided as an identifier using a path descriptor and is used in the XUpdate part to introduce the data at the expected position. The whole process in shown in Figure 3.5.

Note that in the specification, queries are written based on the logical view of the warehouse. Thus, we call them *abstract* queries. The compiler has to translate them into XQuery (or XOQL [11]) queries over the physical XML representation of the warehouse. In a similar way, the (abstract) path identifier of some element (obtained by the `PathIdentifier()` function in a query) has to be translated by the compiler into a call to a function that returns a "real" XPath expression. This XPath expression is based on the specific features of the model, e.g. entity keys. To have XPath expressions that are meaningful in the context of the warehouse, we use our own implementation of the `PathIdentifier()` function.

Example: Consider a Web Service that retrieves the list of authors from the text of a document. The Web Service takes the document (a long text string) as input, and returns a possibly empty list of authors. In the warehouse, we may define the input-query of this service like this:

```
SELECT
<query>
  <input> $a/textcontent
```

```
  </input>
  <location> PathIdentifier($a/URL)
  </location>
</query>
FROM $a in //document
```

This location information is appended to the output of the service call. Thus, the output queries can use this information to place output results in some specific location that is input-dependent. In our example, the result could be:

```
<result>
  <output>
    <author>
      John Smith
    </author>
  </output>
  <location>
    /documents/document[URL:text()="..."]
  </location>
</result>
```

In the example, the Web service is used to detect authors and institutions appearing in the document content. This information is then placed below the document node.

```
SELECT
<xupdate>
  <xupdate:insert path="$a/authors">
    $b
  </xupdate:insert>
</xupdate>
FROM
$a IN result/location::text(),
$b IN result/output
```

This generates the following update script:

```
<xupdate>
  <xupdate:insert
path="//document[URL:text()="..."]/authors">
    <author>John Smith</author>
  </xupdate:insert>
</xupdate>
```

This script is then used to update the warehouse content.

Remark : In some cases, the same Web Service may be called in different places of a warehouse specification. Although it is in principle possible to handle these various calls within a single input query, it is more convenient to define several. Each of them corresponds to some specific use of the Web Service. Output queries are attached to each input query, in order to correctly specify how the results should be processed in each case.

3.4.4 Prototype and experiments

In this section, we briefly present a prototype system and experiments with this prototype within the *e.dot* project. We also mention Web Services that we used.

Prototype A specification language has been defined and a first prototype of a system supporting that language has been implemented. The system provides a graphical editor to specify a warehouse (its data and its services). It also provides a compiler that generates executable warehouse programs in ActiveXML. Finally, while experimenting with "real" warehouses, we developed a set of useful Web Services.

A difference between the specification language in our prototype and the language sketched here is that it uses XOQL [11] instead of XQuery. This is purely because XOQL is at the moment the main query language of ActiveXML. Also, the warehouse is stored in the XOQL persistent store. . We are currently extending the system to use Xyleme Zone Server [162], a large-volume XML repository (that was originally also developed in the group). Input and output queries for Web Services are defined using (i) XPath and XOQL expressions to retrieve nodes, and (ii) predefined XML templates to construct query results. To do so, we had to *package* each Web Service that we plan to use by including some predefined queries and templates. Such packages are essential to facilitate the use of Web Services by warehouse administrators.

Using an early version of the system, we created and monitored a warehouse of INRIA Web sites. We retrieved and stored 20000 pages.

The warehouse for the *e.dot* project that was mentioned is in its early design phase. To experiment, we used a crawl by Xyleme of over a billion documents, from which we selected a list of 22.000 interesting URLs (believed to be) on food risk. A clustering algorithm, provided by the Thesus Web Service [141], with a linear behavior in the number of documents, was able to cluster those 22.000 documents in less than 7 hours. The clustering algorithm is also incremental, and so the connection between the crawler and clustering module should scale to longer duration crawls.

Useful Web Services The core of the processing is done by Web services. Throughout the evolution of the Spin project, we developed a library of Web Services, to be used in regular Web Warehousing applications. We present them here, as some perform the basic services any Web Warehouse administrator will need, for instance to retrieve pages from the Web. In our current project, we also integrated some more advanced services, such as Thesus, to prove the feasibility of integrating complex calculations into the warehouse.

Specific Contributions To find and retrieve data from the Web, we adapted a crawler and a notification/subscription system, developed in the context of the Xyleme project [123, 109, 162] which is described in Chapter 4.

We use a classification and clustering algorithm [76]. The clustering turned out useful in particular to filter out uninteresting pages. It is interesting to note that the Thesus classification algorithm uses yet another Web Service, completely independent of the Spin system, Google's Web Service API [71] that is available on the Web in order to find information such as the incoming links of a page.

We also turned some change control tools [48, 47] into Web services to monitor and version documents in the warehouse. This service is currently used with the biologist's relational data that has been translated into XML format.

More work on some cleaning, XML-izing and enriching of HTML documents is on-going. In particular, we are interested in tagging particular pieces of HTML documents, which involves wrapping them first.

3.5 Conclusion

In this chapter, we have given two examples of large scale Web Warehouses. We have compared their architecture and proposed a specification language and tools to design and construct generic warehouses of Web data. Our system relies on the use of XML for modeling information and Web Services for acquisition or enrichment processing. In our current implementation, we use services that we implemented (e.g. Crawler), as well as services already available on the Web (e.g. Google). The larger and larger number of services available on the Web leads us to believe that the construction of warehouses following our approach will become simpler and simpler, since most functionalities will already be provided as Web services.

Our current implementation is based on ActiveXML that roughly consists of XML documents with embedded service calls. In the future, we are planning to extend our system in different ways: We plan to improve our current Web Services and implement new ones. We are working on the direct compilation of warehouses in Java, and we are also interested in showing how to design and future work involves constructing warehouses of Web data in a peer-to-peer environment.

Chapter 4

Monitoring in Xyleme

Related Publications: Xyleme Monitoring [109, 108].

Strange fascination, fascinating me
Changes are taking the pace I'm going through.
Turn and face the strange
Ch..ch..changes.

David Bowie, Changes

4.1 Introduction

The subscription language we propose is tailored to XML and HTML and offers the possibility to specify complex monitoring and continuous queries with interactions between them. It also provides means of specifying the nature and time of delivery of subscription reports.

Another contribution is the presentation of the general architecture of our subscription system. A major issue in this context is *scalability*. Indeed, most of the problems we consider here would be relatively easy at the level of an intranet. This is not the case when confronted with the sheer size and complexity of the Web. Our monitoring system is designed (and has been tested) to monitor the fetching of millions of XML and HTML documents per day, while supporting millions of subscriptions. The Xyleme system, and in particular, the subscription system we describe here, have been implemented and tested.

The core of the monitoring system consists in what we call the *monitoring query processor* that receives the alerts detected for each document and tests whether they correspond to one or more subscriptions. Towards this end, we propose a new algorithm. We would like to stress that this part of the system

can be used in a much larger setting. In general terms, each alert consists of a set of atomic events and the problem can be stated as finding in a flow of sets of atomic events, the sets that satisfy a conjunction of properties. Our algorithm was designed to support a flow of millions of alerts per day and millions of such conjunctions, and we prove this with experimentation.

The monitoring system we describe here is in use in the commercial system Xyleme (after of course a certain number of modifications). It is therefore used in a certain number of real applications. I was specifically in charge of the design of the architecture of the prototype system, and the implementation of a prototype system. The experiment were run with the help of Jeremy Jouglet. The change monitoring algorithm is joint work with Mihai Preda. I proceeded to its analysis. Gregory Cobena was in charge of the Alerters.

Organization The organization of this chapter is as follows. In Section 4.2, we present a quick review of the subscription system's functionalities. Then we give in Section 4.3 a general overview of the architecture. In Section 4.4, we discuss the Monitoring Query Processor that forms the core of the system, and in particular the algorithm it is based on. In Section 4.5.2 we present a formal complexity analysis of the algorithm. Then, we describe the parts that are specific to the application, the subscription language in Section 4.6 and the alerters in Section 4.7.

4.2 Monitoring Motivations

The subscription language will be detailed in Section 4.6. The role of this section is to give a flavor of the possibilities of the system and thereby motivate the work.

Web users are not only interested in the current values of documents. They are often interested in their evolution. They want to be notified of certain changes to the Web. They also want to see changes presented as information that can be consulted or queried. The main component in the control of changes in such a system are Monitoring queries: Changes to a page are discovered when the system reads the page. The available information consists of the new page, some meta-data such as its type or the last date of update and eventually the signature of the old page or the old page itself (if the page is warehoused). Continuous Queries, i.e., asking the same query regularly in order to discover changes in the answer are also supported by the system, but we made no specific work on the topic.

A subscription consists of a certain number of monitoring queries and continuous queries. Let us consider more precisely monitoring first. The Xyleme system reads a stream of Web pages. We can abstractly view this stream as an infinite list of documents: $\mathcal{D} = \{d_i \mid i\}$, i.e., the list of pages fetched by Xyleme in the order they are fetched. The main task that is set before the change monitoring system is to find, for each document entering the system, if there is a monitoring query interested in this document. If this is the case, the monitoring system produces a *notification* that consists of the code of the monitoring query and some relevant information about the particular document. Thus, each monitoring query can be viewed as a filtering over the list $\mathcal{D}$ of documents. It produces a stream of notifications.

The role of continuous queries is different. Each continuous query is regularly evaluated. Thus the continuous query processor produces a stream of pairs consisting of the code of a query together with the result of the evaluation of this query. By analogy, we also call such a pair, a notification, as shown in figure 4.1. When a particular condition holds, the current value of this stream of notifications is used to produce a (subscription) report using a report query.

There is a last component in the specification of a subscription, namely *refresh statements*. A refresh statement gives the means to a user to explicitly specify that some page or a set of pages has to be read regularly, i.e., to influence the crawling strategy. In our current implementation, subscriptions influence the refreshing of pages only by adding importance to the pages they explicitly mention [103]. Such pages will be read more often.

Thus, a *subscription* more precisely consists of (see Figure 4.1):

1. zero or more *monitoring queries* over the input stream that filter the stream of documents and produce notifications that feed the unique notification stream.

2. zero or more *continuous queries* over the warehouse that also feed the notification stream. Each continuous query comes with a condition to specify when to issue the query, i.e., a frequency (e.g., weekly) or a particular notification.

3. refresh statements that indicate, for instance, that pages for a particular site should be visited at least weekly.

4. an indication, called the *report condition* that specifies when to produce a report and a query, called *report query*, over the notification stream that produces the *subscription report* that is (for instance) emailed to the subscribers.

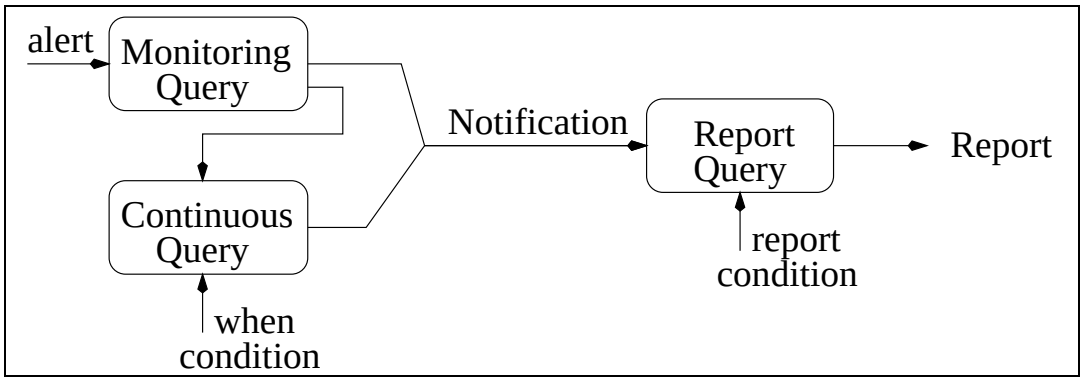


Figure 4.1: Main Components of the Subscription System

To illustrate, an example of a simple subscription using our subscription language is as follows:

```
subscription MyXyleme

monitoring
  select <UpdatedPage url=URL/>
  where URL extends ‘‘http://inria.fr/Xy/’’
        and modified self

monitoring
  select X
  from self//Member X
  where  URL = ‘‘http://inria.fr/Xy/members.xml’’
         and new X

continuous ReferenceXyleme
  % a query Q that computes, e.g., the list of
  % sites that reference Xyleme
  try biweekly

refresh ‘‘http://inria.fr/Xy/members.xml’’ weekly

report
  % an XML query Q’ on the ouput stream
  ...
  when notifications.count > 100
```

This subscription will monitor the set of URLs with a certain prefix that are found to have been modified (since the last fetch) and the new elements of tag *Member* in a particular XML document. It also requests to evaluate query Q (omitted here) biweekly. All the resulting notifications are (logically) bufferized until the report condition becomes true, i.e., until more than 100 notifications are gathered. Once this is the case, a reporting query over all the data gathered so far is issued. The reporting query (also omitted here) may, for instance, remove duplicates URL's of pages that have been found updated several times.

For instance, the previous subscription could return:

```
<Report>
<UpdatedPage url=‘‘http://inria.fr/Xy/index.html’’/>
<UpdatedPage url=‘‘http://inria.fr/Xy/members.xml’’/>
<Member><name>jouglet</name><fn>jeremie</fn></Member>
<Member><name>nguyen</name><fn>benjamin</fn></Member>
<Member><name>preda</name><fn>mihai</fn></Member>
  ...
<ReferenceXyleme>
<site url=‘‘http://www.yahoo.com’’>
<site url=‘‘http://www.amazone.com’’>
...
</ReferenceXyleme>
</Report>
```

4.3 The subscription system

In this section, we present the general architecture of the subscription system shown in Figure 4.2. This architecture can be broken down into two groups of modules.

- Some generic modules that can be used in a more general setting of change control. These are the Monitoring Query Processor, the Subscription Manager, the Trigger Engine, and the Reporter.

- Some application specific modules that are dedicated to the control of change in the Xyleme environment. These include the specific Alerters we are using, the Xyleme Query Processor, and some modules to input subscriptions (Xyleme Subscription Manager) and send results (Xyleme Reporter).

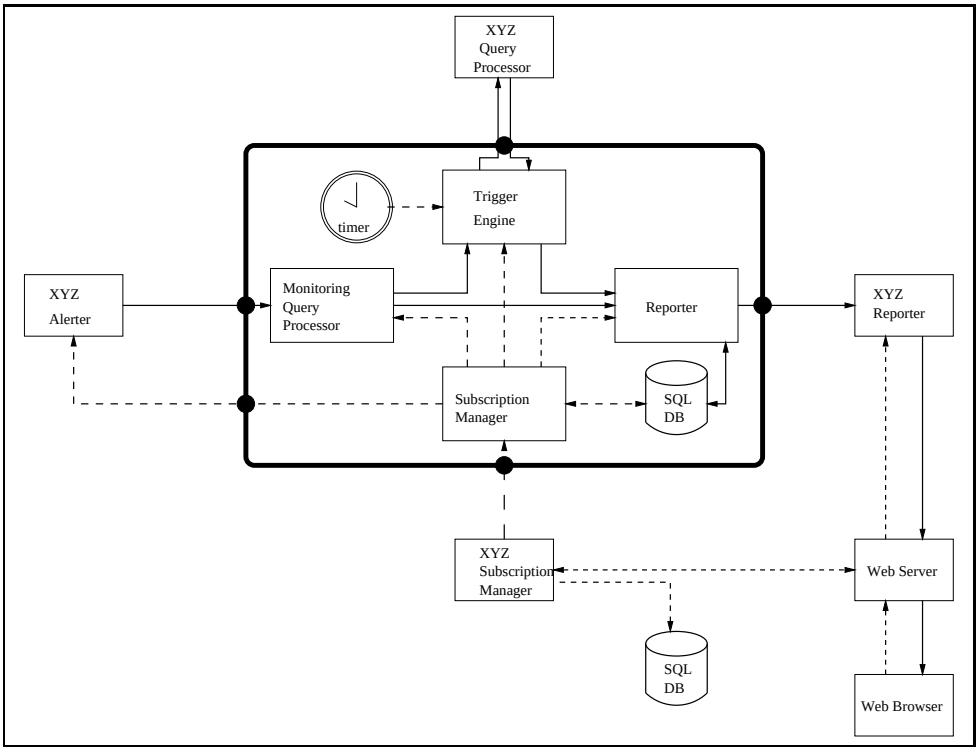


Figure 4.2: Architecture for the subscription system

In the figure, the dotted lines are used for the flow of commands and the filled lines for the data flow. The generic part of the system is within the thick line rectangle.

The global system For each document, the Alerters detect the set of atomic events of interest. If the set is nonempty, an alert is sent to the Monitoring Query Processor that consists of the set of atomic events detected together with the requested data. The Monitoring Query Processor determines whether some subscriptions are concerned with these alerts or whether they should trigger some particular processing. For instance, the Trigger Engine may start the evaluation of a query. Notifications coming from the Monitoring Query Processor (for monitoring queries) or the Trigger Engine (for continuous queries) are sent to the Reporter. When some condition holds, the Reporter sends the set of notifications received so far, an XML document, to the Xyleme Reporter that post-processes it by applying an XML query to it. This produces a report that is either sent by email, or consulted on the Web, with a browser. The Subscription Manager is in charge of controlling the entire process.

The various modules and their roles are considered next.

Alerters (see Section 4.7) The whole monitoring is based upon the detection of *atomic events*. These depend on the type of documents that are being processed. For instance, for HTML documents, typical atomic events we consider are the matching of the URL of the document against some string pattern or the fact that the document contains a given keyword. For XML documents, we would also consider, for instance, the fact that the DTD of the document is a DTD we are interested in, or that it contains a specific tag, or that a new element with a tag we are monitoring has been inserted in the document.

The role of the Alerters is to detect these events for each document entering the system, and if that is the case, to send an alert for the particular document. Thus we see that these modules are *application dependent*. We distinguish between three kinds of alerters: (i) URL alerters that handle alerts concerning some general information such as the URL of a document or the date of the last update, (ii) XML alerters that are specific to XML documents and (iii) HTML alerters for HTML documents. (Only the first two have been implemented.)

Monitoring Query Processor (see Section 4.4) The system must detect conjunctions of atomic events that correspond to subscriptions. We call

complex event such a conjunction of *atomic events*. The role of the Monitoring Query Processor is, based on the alerts raised by a document (i.e. a set of atomic events), the detection of the complex events that this document matches. When such a complex event is detected, the *Monitoring Query Processor* sends a *notification* that consists of the code of the complex event[1] along with some additional data (see the *select* clause further) to the *Reporter* and/or the *Trigger Engine*.

Trigger Engine The *Trigger Engine* can trigger an external action either upon receiving a notification, or at a given date. In our setting, it is in charge of evaluating the continuous queries either when a particular notification is detected or regularly (e.g., biweekly). The query code combined with the result of the query forms a notification that is sent to the Reporter.

The (Xyleme) Reporter The *Reporter* stores the *notifications* it receives. When a report condition is satisfied, it sends these notifications as an XML document. The Xyleme Reporter post-processes this report, basically by applying an XML query to it. Reports are for the moment sent by email. We are considering the support of an access to reports via Web publication which seems more appropriate for very large reports. The main difficulty for the reporter is the management of a heavy load of emails. In our implementation, the Reporter supports hundreds of thousands of emails per day on a single PC. This limitation is due to the UNIX send-mail daemon implementation.

On a single PC, the subscription system can process over 2.4 million notifications per day when connected to the rest of the Xyleme system and hundreds of thousands of emails. It is designed to be distributed although distribution has not been tested yet.

The (Xyleme) Subscription Manager The *Subscription Manager* receives the user requests and manages the other modules of the subscription system. Indeed, the *Subscription Manager* guides and controls the activity of the other modules.

To construct a subscription, a user posts it to an Apache [17] server via a Web browser. The form is subsequently processed by the *Subscription Manager*. Its first role is to serve as an interface for the insertion of new subscriptions and the deletion or modification of existing ones. To be more precise, the Subscription Manager is split into a generic module and an

[1] In fact all the complex events are detected on a document simultaneously and thus are sent to the *Reporter/Trigger Engine* in one batch.

Xyleme specific module. The specific one is in particular in charge of parsing the subscriptions. In our implementation, both modules use the same MySQL database [22] for recovery. Information about users such as email addresses is also stored in this database.

The second role of the manager is to control the various components of the subscription system. For instance, it chooses the internal codes of atomic events and (dynamically) warns the *Alerters* of the creation of new events, their codes and semantic, i.e. the Atomic Condition they correspond to. It will for instance say that Complex Event number 100 is made up of the conjunction of Atomic Events 1 which maps to the Atomic Condition : "The document is an XML document", Atomic Event 5, which maps to Atomic Condition "The document comes from a page whose URL begins with 'http://www-rocq.inria.fr/'" and Atomic Event 434, which maps to Atomic Condition "The document contains the tag 'Xyeleme'". It controls in a similar manner the *Monitoring Query Processor* for managing complex events, the Trigger Engine for continuous queries and the Reporter(s) for reports.

The *Subscription Manager*'s task is not as intensive as that of other modules, since it only depends on the number of people that decide to subscribe to our system at the same time (a few hundred), whereas the *Alerters* depend on the total number of people that have subscriptions (millions). The *Subscription Manager* runs on a single machine.

4.4 Monitoring Query Processor

In this section, we consider the *Monitoring Query Processor*. In particular, we introduce a novel algorithm to perform the detection of subscriptions that are matched by each document. The main difficulty is the heavy rate of arrival of documents and the number of subscriptions we want to process. We next present the problem, the algorithm, then a brief analysis.

4.4.1 Overview

For each document d_i that is fetched by the system, the alerters detect the set of atomic events that are raised by this document. The role of the Monitoring Query Processor is to decide whether this set of events contains all the events in a complex event associated to a particular monitoring query. In such a case, a notification is triggered.

The main difficulty of the problem is that we need to support a rate of millions of documents per day. So, for instance, we cannot afford one disk

I/O access per event detected. Before focusing on the algorithm, we briefly consider without going into details other important aspects of this module:

- The *Monitoring Query Processor* has no semantic knowledge of the data associated to the atomic or complex events it handles. Such additional information is passed in XML format, from the Alerters to the Reporter in a transparent manner.
- Subscriptions keep being added, removed and updated while the system is running. Thus the data structure we use has to be updatable dynamically. Although our system *does* support such updates, we will ignore this issue here.
- Persistency and recovery are handled here by the *Subscription Manager* that pilots a MySQL database.

We now present the algorithm. Let $\{d_i\}$ be the list of documents that is being filtered. Let $\mathcal{A}$ denote the set of all possible *atomic events* where an atomic event in our setting corresponds to an atomic condition in the *where* clause of a monitoring query (see Section 4.6). A *complex event* C is a finite subset of $\mathcal{A}$. The core notification process can be abstracted as follows:

- Let $\mathcal{C} = \{C_j \mid 1 \leq j \leq M\}$ represent the finite set of complex events that is being monitored. (M is the number of complex events.)
- Let $A = \cup\{C_j\}$ represent the finite set of atomic events of interest.
- Let $N = |A|$ represent the total number of atomic events.
- Let $A_i \subseteq A$ represent the set of atomic events detected for document d_i.

The Monitoring Query Processor must determine for each i, the set $\{j \mid C_j \subseteq A_i\}$.

It is convenient to assume some ordering on the atomic events, i.e., $A = \{a_1, ..., a_N\}$. Thus, each A_i and C_j are considered as ordered subsets of A.

Typically, we are considering the case where we have about 1 to 10 million atomic events ($N \approx 10^6$) and 1 to 10 million complex events ($M \approx 10^6$). Note that the problem can be stated as a finite state automata problem. For each i, we need to find the words in $\{C_1, ..., C_M\}$ "contained" in the word A_i. In principle, we could detect this using a finite state automaton in linear time (in the cardinality of A_i) and in constant time in the other inputs to the problem. Unfortunately, because of the size of the problem, the number of states of the automaton would be prohibitive.

We next present the algorithm. It should be noted that before selecting this particular algorithm, we considered alternatives. We found out that the choice of one over the other depended on the conditions of use of the system. A critical factor is the number of atomic events in a complex event. Another critical factor is the number of complex events interested in a specific atomic event. An interesting candidate algorithm we considered turned out to be exponential in that factor. The cardinality of the set of atomic events detected on a document is also important. The algorithm we introduce in the next section presents (as we shall see) a nice behavior with respect to these three aspects as well as others.

4.5 Algorithm: "Atomic Event Sets"

General problem The problem we consider can be stated in the following general terms. Let Γ be a set of sets of atomic events. The elements are called atomic events and sets of elements are called complex events. For each set S of elements, we want to find which are the $C' \in \Gamma$ contained in S. We would like to do that with reasonable time and space costs for a wide range of the parameters of the problem that we will discuss further.

What is important to consider is the context of this application. Typically, we have millions of atomic events, millions of complex events composed of a rather small (4-10) number of atomic events, and thousands of documents per second, that each raise roughly 50-100 atomic events. We analyze here the novel algorithm [109] that, using little space, efficiently detects the complex events triggered by each incoming document. Indeed, the implementation of the algorithm described in [109] is actually in production by Xyleme.

The algorithm The algorithm uses a structure called hash-tree. It is a tree where each node is a hash table pointing to the children of the node. We use some ordering of atomic events. Intuitively, during a pre-processing phase, each complex event in Γ is sorted and entered in the structure. To test a set S of elements, we sort it and we navigate through the structure. Because of hashing, the navigation is very fast.

Note that the data structure we use resembles hash-trees from data-mining [10]. In data mining terms, our problem can be stated as the detection of particular item-sets contained in given transactions. We are unaware of any thorough analysis of the kind presented here.

Contribution Our main contribution, that stems from our collaboration with Mihai Preda, is the proposition of an efficient algorithm to solve this

problem, and an average case complexity analysis of the algorithm. It shows that, within some reasonable ranges of parameters of the problem, a given S can be processed on average in $\mathcal{O}(|S|^2)$. We also present experiments that confirm this theoretical result. Note that this shows, in particular, that (within this range) the complexity is on average independent of the size of the complex events and even of the number of atomic or complex events. We also discuss what happens for the algorithm outside this range.

We will first present the formal problem, in Subsection 4.5.1, and in Subsection 4.5.2, the algorithm. The average case complexity analysis is the topic of Subsection 4.5.3. Experiments are presented in Subsection 4.5.4.

4.5.1 The problem

We use the notation $\binom{N}{k}$ to represent the generalized binomial coefficient (the number of distinct ways to select k elements in a set of N elements):

$$\binom{N}{k} = \begin{cases} \frac{N!}{k!(N-k)!} & \text{if } N > k > 0 \\ 1 & \text{if } k = 0 \text{ or } k = N \\ 0 & \text{if } k < 0 \text{ or } k > N \end{cases}$$

We use lower case letters to represent elements, upper case letters to represent sets, and Greek capitals to represent sets of sets. N, M, h, ℓ represent integers. Let us formalize the problem:

- Let $A = \{a_i | i \in [1; N]\}$ be the finite set of *atomic events*, and $N = |A|$.
- We call *complex event* a subset of A. To simplify the complexity calculation, we will assume at times that all complex events have the same cardinality h. (We will consider the impact of the variations of h).
- Let $\Gamma = \{C_j \mid j \in [1; M]\}$ represent a finite set of complex events. (M is the number of complex events.) We will assume that $A = \cup\{C_j\}$. It is easy to see that $M < 2^N$ and if all complex events are of same length $M \leq \binom{N}{h}$. This is important to bear in mind[2].
- Let $S = \{s_p \mid p \in [1; \ell], s_p \in A\}$ represent a subset of A. The set S is called the set of *incoming atomic events*. Let $\ell = |S|$ represent the number of incoming atomic events.

[2]If the complex events are not all of the same length, and if the longest event has length h_{max} then $M < \sum_{i=1}^{h_{max}} \binom{N}{h} < 2^N$.

Our problem is simple. Let A and Γ be fixed. For a given S, we seek to determine all the $C_j \subseteq S$, i.e., $\{j \mid C_j \subseteq S\}$. We want an efficient algorithm to solve this problem even when N and M are quite large, say order of millions.

Complexity of a "naïve" solution

We first consider the complexity as a function of M, h and ℓ. Let h and ℓ be fixed. The input of the algorithm is Γ and S, where each complex event in Γ consists of h atomic events and S consists of ℓ atomic events.

The output of our algorithm will be the set $R = \{j \mid C_j \subseteq S\}$. First, we do not consider any preprocessing. We suppose that $S \subseteq \cup\{C_j\}$. Let us consider the following (naïve) algorithm.

Algorithm For each set S of atomic events, Γ set of sets of atomic events, the result set R is $Result(S, \Gamma)$ where the (iterative) function *Result* is defined as follows:

FUNCTION $Result(S, \Gamma)$

1. $R \leftarrow \emptyset$
2. FOR every set $C_j \in \Gamma$
   (a) DO
   (b) $B \leftarrow TRUE$
   (c) FOR every $a \in C_j$
      i. DO
      ii. LOOK UP a in S
      iii. IF it is NOT found
      iv. THEN $B \leftarrow FALSE$ ENDIF
      v. DONE
   (d) IF B = TRUE
   (e) THEN add j to R ENDIF
   (f) DONE
3. RETURN R

The lookup of an element in the list can be done in $\mathcal{O}(1)$, using a hash-table for instance. Therefore, the time complexity of this algorithm is $\mathcal{O}(M \times h)$.

Lemma 4.5.1 [Naïve complexity] $c_{naive} = \mathcal{O}(M \times h)$

Let us recall the figures given in the introduction: $\ell \approx 50, h \approx 4, M \approx 10^7$. This "naïve" solution involves a crucial dependency in M that we *want to avoid*. We will see that our algorithm is indeed *independent* of M and N in a broad enough spectrum to be used for our applications.

Summary We propose an algorithm, based on the use of hash-trees, whose average complexity, in a large range of values for N, M, ℓ, h is $\mathcal{O}(\ell^2)$. We present both the complexity calculation, and experimental results that confirm our calculations. Note that in the context we are interested in, h, ℓ are orders of magnitude smaller than M, thus the average complexity is much better that the naïve one and in particular, does not depend on M.

4.5.2 The algorithm

For the purpose of describing the algorithm, we assume that a complex event is a *sorted list* of atomic events[3]. The ordering on the atomic events is purely arbitrary. We also use a data structure in the spirit of hash trees.

The data structure

Observe Figure 4.3. The data structure is entered via table H. This is what we call the ground table. Each other table corresponds to a prefix of complex events, e.g., $H_{1,5}$ corresponds to the complex events starting with $a_1 a_5$. A mark C_j in the corner of a cell indicates that the word that lead to this cell corresponds to complex event C_j. More precisely, a mark C_k in an a_i cell of H indicates that the k-th complex event is the set consisting simply of a_i. Similarly, a mark C_k in an a_i cell of table $H_{i_1 \ldots i_k}$ indicates that the C_k complex event is $a_{i_1} \ldots a_{i_k} a_i$. For instance, if you look at Figure 4.3, you see that cell a_0 in table H is marked with the value C_0. This means that the complex event $C_0 = \{a_0\}$. In the same way, cell a_6 of table $H_{1,5}$ is marked with value C_{43}. This means $C_{43} = \{a_1, a_5, a_6\}$. However, cell a_5 in table H_1 is not marked. This means the set $\{a_1, a_5\}$ does not constitute a complex event.

We consider that each table is stored as a *hash-table*. This makes every look-up in the table cost $\mathcal{O}(1)$. It is noteworthy that the size of the structure is small enough to reside in main memory in our experiments. This size is less or equal to Mh. The upper limit is met in the case where no complex

[3] This can be trivially done in $\mathcal{O}(\ell \log \ell)$ so does not change the final complexity result.

events share the same nonempty prefix. Finally, such a data-structure can easily be updated by adding or deleting complex events.

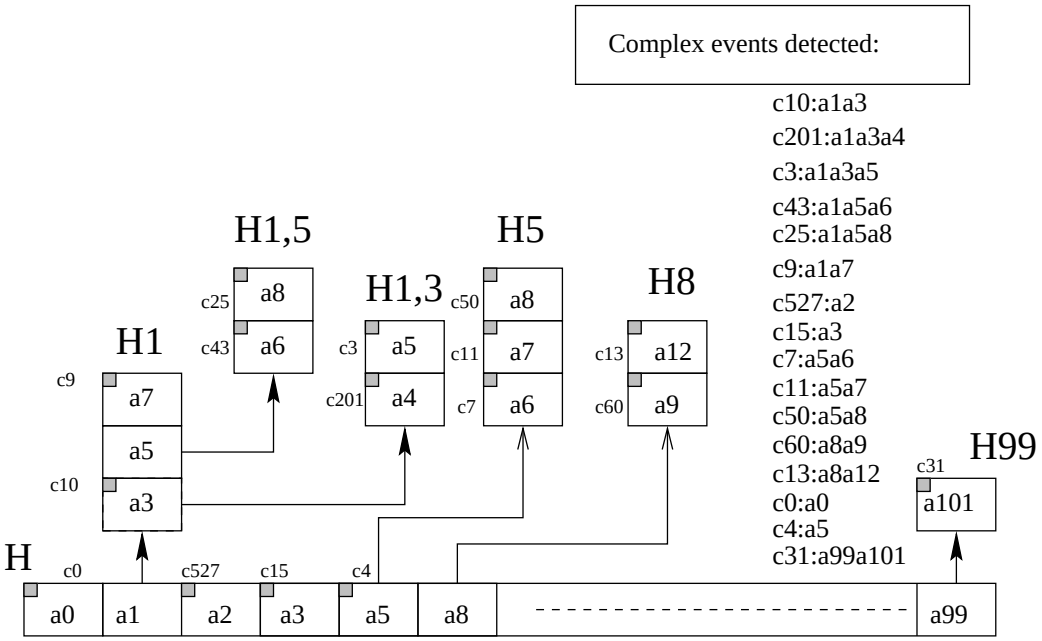


Figure 4.3: The data structure with complex events of variable length

Example

Let us demonstrate how the algorithm runs using an example. Suppose the document that is being processed has triggered the ordered atomic event set $S = \{a_1, a_3, a_5\}$. We first enter the data structure by $H[a_1]$, i.e., the first event of S. (Since the box is not marked, a_1 alone is not a complex event.) We proceed to H_1. We find a_3. Since the box is marked, we detected the complex event $C_{10} = \{a_1, a_3\}$. Then we proceed to table $H_{1,3}$. We see that the cell containing a_5 is marked, this means we also detect the complex event $C_3 = \{a_1, a_3, a_5\}$. Back in table H_1 we find a_5, but the cell is not marked. Since we have no more atomic events left to process in this sub table, we exit it. We now reenter the data structure with a_3, to find $C_{15} = \{a_3\}$. Since this cell does not point to a sub table, we stop its processing here. Then we enter cell a_5 and detect the complex event $C_4 = \{a_5\}$. We have no more atomic events to process, so our task is over. The document triggered 4 complex events: C_{10}, C_3, C_{15} and C_4.

Presentation of the algorithm

For each table T (which is a hash-table) and each atomic event a_k, we denote by $T[a_k]$ the a_k entry of the table. Starting from the *ordered set* S of incoming atomic events, we produce the set of complex events that are triggered using the following algorithm:

Algorithm For each ordered set S of atomic events, the result set R is processed by*Notif*(H, S) where the (recursive) function *Notif* is defined as follows:

T is the H table or an H_ω table and

$a_{i_1}...a_{i_n}$ is an ordered set of atomic events.

FUNCTION *Notif*(T,$a_{i_1}...a_{i_n}$)

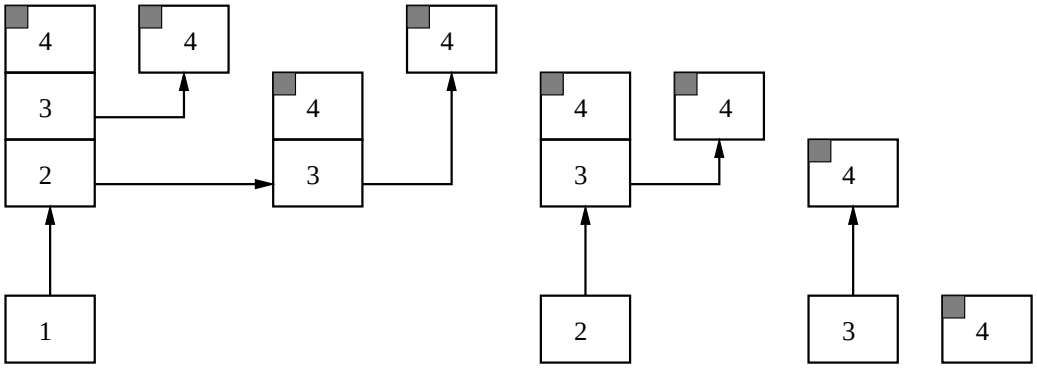
1. $R \leftarrow \emptyset$
2. FOR each k in $[1..n]$
   (a) DO
   (b) IF $T[a_{i_k}]$ is marked THEN add its mark to R ENDIF
   (c) IF $T[a_{i_k}]$ points to some table T' THEN add *Notif*($T', a_{i_{k+1}}...a_{i_n}$) to R ENDIF
   (d) DONE
3. RETURN R

To obtain the complex events detected by a set S we simply call $Notif(H, S)$. Let us recall that since we use hash-tables, the costs of table lookups in 2-b, 2-c, is $\mathcal{O}(1)$.

We will briefly consider in the following paragraphs the worst case complexity of our algorithm, considering two parameters, the cardinality ℓ of S and h of each C_j.

Worst case complexity in ℓ

In the worst case, for every subset of S we look for, we find an entry in the table. Thus, the number of lookups in the table is $C_{wc_1} = \mathcal{O}(2^\ell)$. For example, this situation arises for the instance of the problem depicted in Figure 4.4.



S={1,2,3,4}

Figure 4.4: Worst case complexity example

Worst case complexity in h

The cardinality h of each C_j acts as a cut-off point for the worst case complexity. To obtain a complexity in $\mathcal{O}(2^\ell)$, we need to explore the data structure up to level ℓ. This supposes that $\ell < h$. On the contrary, if $h < \ell$ then we will stop at level h in the worst case. Thus:

$$C_{wc_2} = \mathcal{O}(\ell^h)$$

This is a more realistic bound in the context of the applications that motivated the present work, where $h \approx 5$ and $\ell \approx 100$.

4.5.3 Average case analysis

In this section, we study the influence of the length ℓ of the incoming subset S. To simplify the analysis, we suppose in what follows that all complex events are of length h. We show results on the average case time complexity of the algorithm.

Preamble

We assume that the ℓ atomic events in S have been uniformly randomly selected. Recall that the tables in the data structure are stored as hash-tables, and that S is stored as a list. Since a look up in a hash table is an $\mathcal{O}(1)$ action, the lookup time of a sublist of ℓ_i elements of S has a complexity $\mathcal{O}(\ell_i)$. The four critical parameters in the problem are N, M, ℓ, h. We found that a direct analysis based on N, M, ℓ, h was too complex. Instead, we

present an analysis of the algorithm by focusing on the influence of the ℓ parameter. We will revisit the influence of N, M and h at the end of this section.

We assume that the time complexity of the algorithm is composed of the sum of the time spent making the look-ups in the hash tables.

The data structure we use can be seen as made of *levels*. The *ground* level consists of table H that we will call the *ground table*. The first level consists of all tables H_i. More precisely, we call *first level of* a_i the table H_i (that element a_i in table H points to). In the same way, we call *n-th level of* $a_{i1}..a_{in}$ the table $H_{i_1,...,i_n}$. We will see that under rather general assumptions, the cost is dominated by the cost of computing up to the first level since further levels will not be reached most of the time.

We consider separately the cost function computations at each level. This motivates the following definition:

Notation 4.5.1 *Let N, M, h be fixed. The complexity cost function in ℓ is denoted $cost^{N,h,M}(\ell)$. The cost of processing events in the j-th level of the data-structure is denoted $cost_j^{N,h,M}(\ell)$; $cost_0^{N,h,M}(\ell)$ is the cost for level 0, the ground level. When N, h, M are understood, we simply use $cost(\ell)$ and $cost_j(\ell)$.*

Using this notation, $cost(\ell) = \sum_{j=0}^{\ell} cost_j(\ell)$. Observe that, $cost_0(\ell)$ represents the number of look-ups we will make in table H; $cost_1(\ell)$ represents the number of look-ups we will make in tables $H_1, H_2....H_{N-h}$; $cost_2(\ell)$ represents the number of look-ups we will make in tables[4] $H_{1,2}, H_{1,3}....H_{X_1,X_2}$. As we will see later, some of these tables may not exist, so the corresponding look-up cost is null.

We use the following notation:

Notation 4.5.2 *Given h, N, the probability that one (random) complex event of size h starts with a_i is noted: $START_{N,h}(i)$.*

The proof of the following lemma can be found in Appendix A:

Lemma 4.5.2 Given N, h, i, we have:

$$\begin{cases} \text{if } i \leq N-h+1: & \\ \quad START_{N,h}(i) & = \frac{\binom{N-i}{h-1}}{\binom{N}{h}} \\ & = \frac{1}{(h-1)! \times \binom{N}{h}} \prod_{j=0}^{h-2}(N-j-i) \\ \text{if } i > N-h+1: & \\ \quad START_{N,h}(i) & = 0 \end{cases}$$

[4]We cannot enumerate all the tables that we are going to do lookups in, in a general way, since they depend on the complex events that are stored.

and $START_{N,h}(i) = \mathcal{O}(\frac{h}{N}(1-\frac{i}{N})^{h-1})$

For an experimental illustration of this result, see Figure 4.5(a). We plot on those figures on the Y-axis, the number of complex events that start with the atomic event i, where i is on the X-axis. The plot represents the experimental points, i.e. the exact number of complex events beginning with a given atomic event, after generating a sample M complex events. The full line curves represent the theoretical results. On the second row of figures, we plot on a logarithmic y-axis, for better readability. Note that, for $i > N-h$, there cannot be a complex event of length h that starts with a_i. Observe also that for some i, table H_i may be empty. This is the case if each complex event containing a_i also contains some a_j for $j \leq i$. In the conditions of our application, this case does not occur often as we see next. We use the following notation:

Notation 4.5.3 (Existence of table H_i) *Let $LAYER^1_{N,h,M}(i)$ represent the probability that table H_i exists (i.e. is not empty), after randomly generating M complex events.*

For an analysis of the $LAYER$ function, see Appendix B.

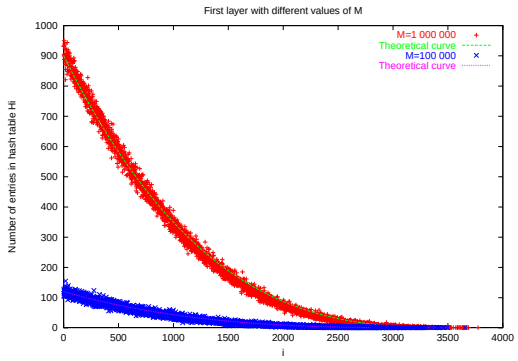
To simplify the analysis of the problem, we will make the *pessimistic* following assumption:

Assumption 4.5.4 (Full ground layer assumption) *For all $i \in [1, N-h]$, $LAYER^1_{N,h,M}(i)$ is approximated by 1.*

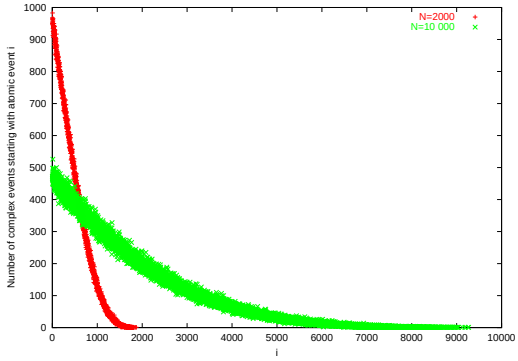
This means we simply ignore the fact that certain entries in the ground table may be empty. This is a pessimistic assumption since in practical cases, some of the tables will be. If we find such a cell pointing to an empty table, the search with the cell as prefix will be over with at no extra cost. However, for the range of N and M we are interested in, the impact is negligible. See Figure 4.5(b) for values of N, M, h for which many H_i tables are empty. For instance, in the case of $N = 10000$ we see that most of the last 1000 tables are empty, but that the first ones contain at least one element.

Computing up to the first layer

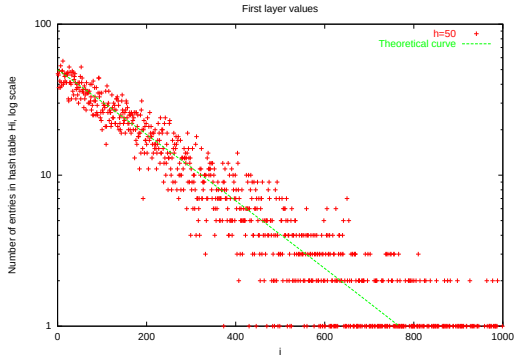
Let us first ignore the cost of further layers. We calculate the value of $cost_0(\ell) + cost_1(\ell)$. The operations that are executed are the following: For each element in the incoming list, we lookup in the first layer all the elements that have an index greater than it in the list, thus for the i-th element of the



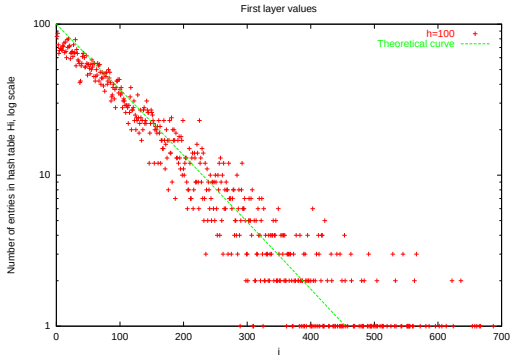
(a) Number of complex events beginning with atomic event a_i for N=4000, h=4, M=100 000, 1 000 000 000



(b) Number of complex events beginning with atomic event a_i for M=1000000,h=4,N=2000,10000



(c) Number of complex events beginning with atomic event a_i for N=10 000, h=50, M=10 000



(d) Number of complex events beginning with atomic event a_i for N=10 000, h=100, M=10 000

Figure 4.5: The experimental results show that the number of events in the first layer is indeed proportional to $M \times START_{N,h}(i)$. Figures 4.5(c),4.5(d) are presented with a logarithmic y-axis, to stress the polynomial fit, according to Lemma 4.5.2. Figure 4.5(a) presents the polynomial curve in the power $h-1$.

incoming list, we will have to lookup $\ell - i$ elements in the first layer table. Assuming that the first layer is full, we have:

$$\begin{aligned} cost_0(\ell) &\leq \ell \times \mathcal{O}(1) \\ cost_1(\ell) &\leq \sum_{j=1}^{\ell-1} j \times \mathcal{O}(1) \\ cost_0(\ell) + cost_1(\ell) &\leq \sum_{j=1}^{\ell} (1+j) \times \mathcal{O}(1) \\ cost_0(\ell) + cost_1(\ell) &= \mathcal{O}(\ell^2) \end{aligned}$$

Computing further...

We will now show that, under reasonable conditions, the contribution of the higher levels is of the same order of cost. We will start with the case of the second layer. We will use the following notation:

Notation 4.5.5 (Existence of table $H_{i,j}$) *Let $LAYER^2_{N,H,M}(i,j)$ represent the probability that table $H_{i,j}$ exists, after randomly generating M complex events.*

The problem is that, of course, this probability is dependent not only of i, the index of the first atomic event, but also of j, the index of the second atomic event. Nevertheless, to calculate an upper bound on the complexity cost of the second layer, we only need to know *how many times* we will enter into *any* hash-table of the second layer. Therefore, we only need to know how many *different* atomic events appear in table H_i, to be able to determine how many times (on average) we are going to find an element in this table, and ultimately, how many times we will have to proceed to the second layer.

If we neglect (pessimistically) the fact that we can generate complex events that share the same first two atomic events, the number of events in hash-table H_i will be smaller than $START_{N,h}(i) \times M$. Therefore, the probability of finding a given element in the first layer will be equal to $LAYER^2_{N,h,M}(i) = \frac{START_{N,h}(i) \times M}{N-i}$. Since we assumed that all the tables H_i exist, for all the ℓ elements, we will try all the others greater than them (once again ℓ is clearly an upper bound), and if they are found in the table, pay an extra cost for the second layer[5], that is clearly $\mathcal{O}(\ell)$. Thus we have an upper

[5]The cost to look up a maximum of ℓ elements in the next layer. Of course, there will in fact be less than ℓ elements, since at each level, all the elements in S that have an index smaller to the one processed will have been set aside

bound for the cost of the second layer:

$$cost_2(\ell) \leq \ell^3 \times \frac{START_{N,h}(i) \times M}{N-i} \times \mathcal{O}(1)$$

We prove in Appendix C the following lemma:

Lemma 4.5.3 $\forall (N,h,i) \in (\mathbb{N}^*)^3, if\ (h+i+1) < N$, then $\frac{\binom{N-1}{h-1}}{\binom{N}{h} \times (N-1)} \geq \frac{\binom{N-i}{h-1}}{\binom{N}{h} \times (N-i)}$

This lemma shows that that an upper bound to $cost_2(\ell)$ can be given by

$$cost_2(\ell) \leq \ell^3 \times \frac{START_{N,h}(1) \times M}{N} \times \mathcal{O}(1)$$

To simplify our computation, let us assume the following:

Assumption 4.5.6 (Relative upper bound of M) $M \leq \frac{N^2}{2(\ell-2)h}$.

This is a realistic assumption since, for instance, for $N = 40\ 000, h = 4, \ell = 20$, this yields $M < 10\ 000\ 000$, exactly the kind of bounds we need in our application. The interpretation can be seen as wanting to avoid the case depicted in Figure 4.4, where only a small number of atomic events constitute many complex events.

Under Assumption 4.5.6, we see that

$$\begin{aligned}
cost_2(\ell) &\leq \ell^2 \times START_{N,h}(1) \times \frac{N}{2h} \\
cost_2(\ell) &\leq \ell^2 \times \frac{\binom{N-1}{h-1}}{\binom{N}{h}} \times \frac{N}{2h} \\
cost_2(\ell) &\leq \ell^2 \times \frac{(N-1)! \times h! \times (N-h)!}{(h-1)! \times (N-h)! \times N!} \times \frac{N}{2h} \\
cost_2(\ell) &\leq \ell^2 \times \frac{1}{2}
\end{aligned}$$

We can conduct the same upper bound calculation on the higher levels of the data structure, and in a similar way, produce the following result:

Lemma 4.5.4 [Cost of layer j] Under Assumption 4.5.6,

$$\forall j \geq 1, cost_j(\ell) \leq \ell^2 \times (\frac{1}{2})^{j-1}$$

We see that this gives us an upper bound on the global complexity:

$$\begin{aligned} cost(\ell) &\leq (\sum_{j=1}^{\ell} cost^j_{N,h,M}) + \ell \\ cost(\ell) &\leq \ell + \ell(\ell-1) \times \sum_{j=1}^{\ell} (\frac{1}{2})^j \\ cost(\ell) &\leq \ell + 2\ell(\ell-1) \end{aligned}$$

And so we conclude our average case analysis with the following result:

Theorem 4.5.5 [Average case complexity] Under Assumption 4.5.6,

$$cost_{N,h,M}(\ell) = \mathcal{O}(\ell^2)$$

Experimental measures will confirm this result.

Remark 4.5.6 (Parameter h) As shown in the previous section, if we consider h small, it does not come into account in the calculation of the average complexity, or the lower bound complexity, its only influence is in the worst case.

Remark 4.5.7 (Dependency on M and N.) Each complex event is constructed with atomic events, so given N, we have $N/h \leq M \leq \binom{N}{h}$. We will see experimentally that the algorithm indeed behaves in $\mathcal{O}(\ell^2)$ for a wide range of values. However, it behaves differently at the extremes:

lower end When M is much above the threshold given in our analysis (see Assumption 4.5.6 $M_{cutoff} > \frac{N^2}{2h\ell}$), the cost of the computation in deeper layers than the first one starts dominating. In such cases, we should expect the average complexity to approach the worst case complexity $C_{wc}(h)$. This will be confirmed experimentally.

upper end When M is very small, our analysis is too pessimistic. Most cells we examine are empty. At the limit, the complexity becomes on average independent of ℓ. This will be confirmed experimentally.

4.5.4 Experimental results

Implementation We have implemented the algorithm in C++, using a hash-tree library we developed. It performs very well in the context of our application. As a matter of fact, it is now being used in production in the Xyleme system. We also performed some extensive tests. We would like to stress that the tests were performed on a very standard machine: a Linux PC with 700MHz Pentium III processor and 1GB of RAM.

FUNCTION
  $GenerateComplex(N_0, h)$
$C \leftarrow \emptyset$
$i \leftarrow 0$
WHILE $i < h$ DO
 Randomly select a in $[1..N_0]$
 IF $a \notin C$ THEN
  $C \leftarrow C \cup \{a\}$
  $i \leftarrow i + 1$ ENDIF
 DONE
RETURN C
	FUNCTION
	  $Generate(N_0, M, h)$
	$\Gamma \leftarrow \emptyset$
	$j \leftarrow 0$
	WHILE $j < M$ DO
	 $C \leftarrow GenerateComplex(N_0, h)$
	 IF $C \notin \Gamma$ THEN
	  $\Gamma \leftarrow \Gamma \cup \{C\}$
	  $j \leftarrow j + 1$ ENDIF
	DONE
	RETURN Γ

Figure 4.6: Generation of events algorithm

Random complex event generation We briefly describe the construction of complex events for our tests.

We start by fixing N_0. This represents the universe from which we will be able to select atomic events. We then construct M distinct complex events by calling the following function: $Generate(N_0, M, h)$ given in Figure 4.6.

We see that the algorithm simply generates complex events until it finds one that does not yet belong to the set that has been constructed so far. We stress that we do not control exactly the number N of atomic events used, some integers between 1 and N_0 may not have been selected. In the results we present in this section, we use the exact count of atomic events, i.e., N and not N_0. (During all the experiments, we selected a-priori N_0 so that $N_0 \approx N$.) These functions are used to generate the set M of complex events. To generate the incoming sets of atomic events, we also use the function *GenerateComplex*. To be precise, we also have to test that each atomic event appears in some complex one, i.e., is not in the $N_0 - N$ atomic that were not selected.

Let us discuss two simplifying assumptions:

equiprobability The analysis we made in the previous sections assumes that all atomic events have the same probability of appearing in a complex event. The random generation we use guarantees such a distribution. This has been assumed for the sake of simplicity. In practice, this is far from being the case. Some atomic events are much more likely to occur in complex events than others. It would be interesting to run experiments with a different random event generation, where we would for instance define atomic events by the percentage of complex events they appear in.

fixed length We assumed so far that all complex events are of the same

length. We also ran tests with variable length complex events. First, we determined the length of a complex event, by picking a random value equally distributed between 1 and $2h-1$. Then we randomly constructed a complex event of that length. The experimental results were analogous to the ones presented here.

As already mentioned, we have measured the number of complex events that begin with a given atomic event in a number of cases. Some such measures are given in Figures 4.5(a), 4.5(c), 4.5(b). We see that, as proved in Lemma 4.5.2, this number is proportional to $START_{N,h}(i)$.

Influence of ℓ As shown in Theorem 4.5.5, the complexity of the algorithm, given N and M within certain bounds, is $\mathcal{O}(\ell^2)$.

Param	Value	Asymptotic Std Err
a_2	2.2×10^{-4}	$\pm 4.7 \times 10^{-6}$ (2.1%)
a_1	4.5×10^{-1}	$\pm 4.6 \times 10^{-3}$ (1.0%)
a_0	5.6×10^{-1}	$\pm 6.5 \times 10^{-1}$ (115%)

Table 4.1: Results of the polynomial fit

The experimental results confirm our complexity calculations. We have conducted extensive tests, with values of M and N ranging from 1000 to 10^7, and they all exhibited the same parabolic behavior. To illustrate the experiments, we show in Figure 4.7 the variation of the computational time for the following values: $N = 10^6, M = 10^6, h = 4$ and ℓ ranging from 1 to 1000. Figure 4.7 and Table 4.1 show that the complexity is closely fitted by a curve of the form $f(\ell) = a_2\ell^2 + a_1\ell + a_0$. We plot the results using gnu-plot in Figure 4.7, and we provide the results of the fit in Table 4.1. We see that for an incoming set of 1000 atomic events, the time to process it is 0.5s, which means about 2000 documents per second, approximately 175 million documents per day. This can be compared to the rate of a single crawling machine, which is of a few million documents per day. This means *a single* complex event detection system can process the documents loaded by many different crawlers simultaneously.

Influence of parameters N and M Figures 4.8 show that for N and M ranging from 1000 to 1000000, their influence on the complexity of the algorithm is minimal. We have also conducted experimentations to see the influence on M and N in areas where our assumptions were not valid anymore. As previously mentioned, when $M >> N$, there are more and more

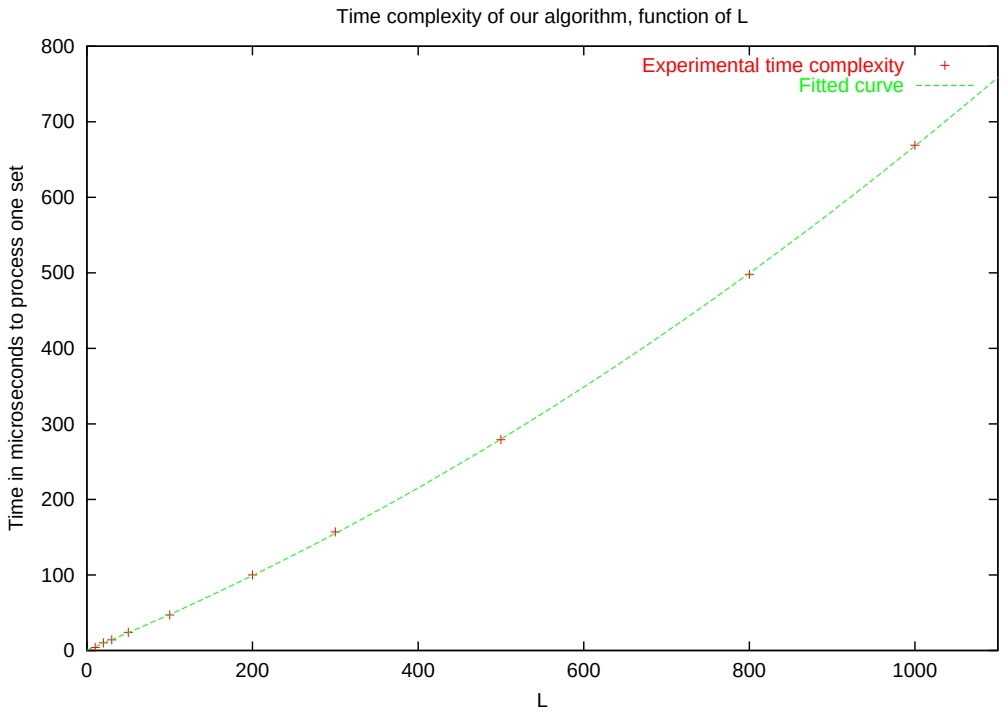
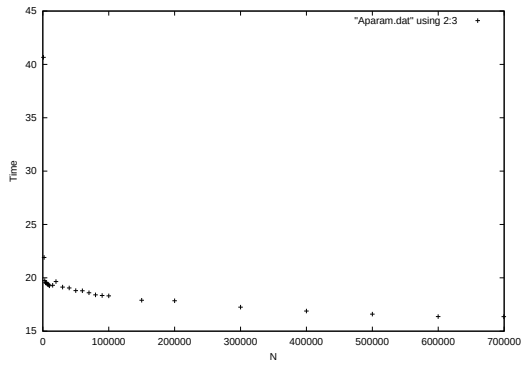
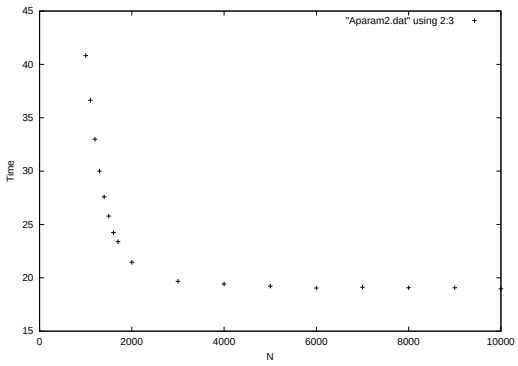


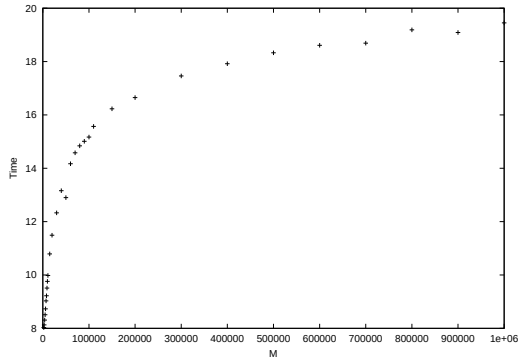
Figure 4.7: Time complexity function of ℓ, with $N = M = 10^6, h = 4$



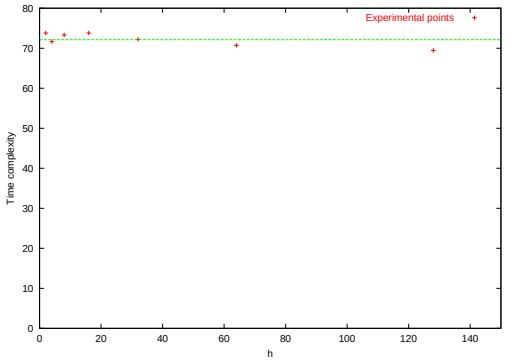
(a) Time complexity in microseconds, function of N for M=1000000,h=4,l=20



(b) Time complexity in microseconds function of N for M=1000000,h=4,l=20 (zoom)



(c) Time complexity in microseconds, function of M for N=100000,h=4,l=20



(d) Time complexity in microseconds, function of h, with M=100000, N=100000, l=50

Figure 4.8: Time complexity function of N, M and h

chances of finding atomic events in lower layer tables, and we are close to the worst case scenario we discussed in Section 4.4. For $M = 10^6$, our analysis is correct for $N_{min} > 12000$. If we look at the zoom in Figure 4.8(b), we see that we in fact our analysis is correct for $N > 2000$. This confirms experimentally Lemma 4.5.2.

Figure 4.8(c) shows that for very few complex events compared to the number of different atomic events, then the performances of the algorithm are even better than the ones we expect. The high cutoff point for M is not shown in Figure 4.8(c), its approximate value is $M_{cutoff} = 10^{12}$. We can also note a 'low' cutoff point for M, for which the algorithm performs even better than expected. This value has not been calculated, but graphically we see it is around $M = 200000$. In between the high and low cutoff points, the time complexity is approximately constant (of M).

Influence of parameter h: We conducted two series of experiments:

influence of the length of h Measurements made with different values of h showed that this parameter does not influence much the cost of the computation. This can be seen in Figure 4.8(d). (Note that for $h > \ell$, it is impossible to trigger a single complex event.)

variable length complex events We also tested this algorithm on a more "realistic" set of complex events, where all did not have the same length. Their length was randomed between 1 and $2h - 1$. The experiments showed that the cost was roughly that obtained when all complex events have the same length h. We did not analyze this problem any further.

Influence of the number of complex events detected: It can be argued that in real world applications the atomic events raised by a document are going to be correlated, and there is a greater chance to trigger at least one complex event for each document. We run tests by skewing the elements of the incoming set of atomic events S by generating S in such a way that it contains at least x complex events. We ran tests with x ranging from 1 to 10 that give the same behavior in $\mathcal{O}(\ell^2)$ as the non skewed ones.

4.5.5 Conclusion

The management of subscriptions in a Web warehouse motivated the study of an algorithm for subset detection whose average time cost would be independent of the number of candidate subsets. The context eliminated solutions

that would require too large amounts of memory. The algorithm we presented and its implementation indeed have satisfying performances for our application. Moreover, our implementation is now being used in the Xyleme system.

Our main result is that, for a very wide range of values of M and N, our algorithm is in $\mathcal{O}(\ell^2)$. We also performed in-depth tests that we discussed here.

In this work, we assumed no a-priori knowledge on the atomic conditions. In practice, we know a lot about them. First, we may have knowledge about their distribution, and they are not equiprobable. For instance, suppose that a_1 is that the document is in English and a_2 that it contains the word "antidisestablishmentarianism". Clearly, a_1 is much more likely to happen. Note also that it is likely that every document raising a_2 would also raise a_1. By taking advantage of this knowledge, one could imagine ordering the atomic events to lower the computational cost.

The algorithm detects conjunctions of atomic events. It would be interesting to consider more complex conditions, *disjunctions* and *negations*. Moreover, in our setting, it would be interesting to study "temporal conditions" such as $a_1; a_2$ is raised by a document and later $a_3; a_4$ but with no a_5 occurring in between.

Measures show that the algorithm can process several thousand sets of atomic events per second on a standard PC. This should be compared to the rate of our crawler. Currently, one Xyleme crawler [103] is able to fetch about 4 million pages per day, that is approximately 50 per second. Thus the *Monitoring Query Processor* we described here can support the load of about 100 crawlers. The data structures we use require about 500MB of memory for $Card(A) = 10^6, Card(C) = 10^7$ and $h = 10$. Like the other modules, the Monitoring Query Processor was designed so that its task could be distributed on several machines. Typically, one can use distribution along two directions:

1. Processing speed: we can split the flow of documents into several partitions and assign a Monitoring Query Processor to each block of the partition.

2. Memory: we can split the subscriptions into several partitions and assign a Monitoring Query Processor to each block. This results in smaller data structures for each processor.

Based on these two kinds of distributions, we obtain a very scalable system.

Observe to conclude this section that the Monitoring Query Processor is generic in that it may be used for a very wide range of applications. We next

turn to parts of the system that are more specific to monitoring a flow of Web pages. In the next section, we consider the subscription language; and in the following one, the specific alerters we use.

4.6 The subscription language

In this section, we briefly describe the subscription language.

As we have seen in Figure 4.1, a subscription consists of the following parts: (i) monitoring queries, (ii) continuous queries, (iii) report specification, and (iv) refresh statements, and is of the following form:

```
subscription name
  monitoring...          % (i)
  continuous...          % (ii)
  report when...         % (iii)
  refresh...             % (iv)
```

We next consider (i,ii,iii) in turn; (iv) has not been implemented. To conclude this section we consider the issue of controlling subscriptions to block requests that would require too many resources.

4.6.1 Monitoring Query

A monitoring query has the general form:

```
select  result
(from   from-clause)
where   condition
```

The *from* clause This clause may be omitted because we know the data that is being filtered, i.e., the document that is currently being processed. This current document is denoted *self* in the query. A *from* clause may be used to attach variables to elements of the current document.

The *select* clause This clause describes the data the resulting notification should contain based on constants, *self* or variables defined in the *from* clause. The notification itself is an XML element. For the moment, we have not implemented this part of the system. Notifications simply return the URL of the document that triggered the monitoring query and basic informations about the document.

The where clause This clause is a condition that consists of a *conjunction* of *atomic conditions*. An atomic condition may be of one of the following:

URL **extends** string	URL = string
DTDID = integer	DTD = string
DOCID = integer	domain = string
filename = string	

where *domain* is one of the semantic domains that Xyleme uses to classify documents (e.g., biology), *DOCID, DTDID* are internal identifiers, and *filename* is the tail of an URL (e.g., index.html) . Other atomic conditions deal with information about documents that either is maintained by Xyleme or can be obtained by the alerters when the document is fetched: e.g.,

- *LastAccessed* ⟨comparator⟩ date
- *LastUpdate* ⟨comparator⟩ date
- *self contains* string
- ⟨*status*⟩ *self*

where ⟨*status*⟩ is defined as one of: *new, unchanged, updated, deleted*[6].

The most interesting atomic conditions deal with element values inside a document and are meaningful only for XML documents. Their syntax is as follows[7]:

(⟨*change*⟩) ⟨*element-name*⟩ (*contains string*)

The ⟨*change*⟩ keyword means some change pattern of the elements of a given name (tag) will be monitored, e.g., we are only interested in documents containing a new element with the tag *product*. We may also impose that the element contains a given string, e.g., we are only interested in document with an element containing the word *electronic* and under the tag *category*. The semantic of *contains* is that this string occurs within the element. We also support *strict contains string* that specifies that the string must be present directly in the text of the element.

A *where* clause is a conjunction of atomic events, e.g.:

- *new self*
  and URL extends "http://www.xyleme.com//"
  (new documents with a certain URL pattern)

[6] We will not discuss deletions here. It is not an obvious notion since deletion is rarely explicit on the Web.

[7] Parenthesis denote an optional parameter

- *new Product*
  and URL extends "http://www.amazon.com/catalog/"
  (documents in a particular catalog containing a new product)

- *updated Product contains "camera"*
  and DTD="http://www.amazon.com/dtd/catalog.dtd"
  (documents with a particular DTD containing an updated product containing the word camera).

Each atomic condition is mapped to an atomic event. Note that it is likely that each document we read will raise one atomic event involved in at least one subscription, i.e., one in *new, unchanged, updated.* So, if we are not careful we would have to raise one alert for each document, that is, we would have to send a set of atomic events to the Monitoring Query Processor for each document. To avoid this, we distinguish between *weak* events (*new*, *modified*, *unchanged*) and *strong* events (all other atomic events). We disallow *where* clauses composed solely of a weak atomic condition. Thus, a document is detected as potentially interesting if at least a strong atomic event of interest for a subscription is detected. In this case only, an *alert*, consisting of the set of atomic events detected plus some extra data (defined by the *select* clause) is sent to the *Monitoring Query Processor.*

4.6.2 Continuous queries

As mentioned in Section 4.2, a subscription may also include one or more continuous queries that will participate in the notification stream. A continuous query consists of a standard Xyleme query [12] plus a condition that specifies when to apply the query. Typically, this condition involves a frequency (e.g., every week). The continuous query may also be triggered by a notification sent by the subscription processor.

Consider the continuous query:

```
continuous delta AmsterdamPaintings
  select p/title
  from   culture/museum m, m/painting p
  where  m/address contains "Amsterdam"
  when   biweekly
```

that asks for the names of all paintings found in an Amsterdam museum. Here *culture* is an *abstract* domain that provides an integrated view of museum resources. We ask the system to evaluate the query twice a week. The report will therefore contain a list of results:

```
<AmsterdamPaintings> ... </AmsterdamPaintings>
<AmsterdamPaintings> ... </AmsterdamPaintings>
```

The use of the keyword *delta* specifies that we are interested by changes to the result and not by the result *per se*, and that we are interested in storing the delta of this document [98]. So the first time the query is evaluated, we get its answer, but later, we only receive the modifications of the result such as:

```
<AmsterdamPaintings-delta> ...
<inserted ID="556" parent="556" position="4">
     <title> ... </title>
     <title> ... </title>
<updated ID="332" note="not available
  -- visiting MOMA">
</AmsterdamPaintings-delta>
```

Identifiers such as 556 are used here as the foundation of a naming scheme for XML documents that makes the specification of changes easier. *Deltas* based on XIDs provide a compact naming of the elements of the documents that is the basis of the versioning mechanism of the system. In particular, the new version of a document can be constructed based on an old version and the delta. We also provide a practical change editor for the visualization of changes in XML documents or query results in the spirit of change editors as found, for instance, in MS-Word. A key component for defining these changes is the *diff* package for XML that we developed. A detailed presentation of these mechanisms can be found in [98].

In the previous example, the continuous query is asked with some time frequency. It is also possible to trigger the evaluation of a continuous query with notifications from a monitoring query as in:

```
subscription XylemeCompetitors

monitoring
    select <ChangeInMyProducts/>
    where URL = ``www.xyleme.com/products.xml''
          and modified self

continuous MyCompetitors
    select ...
    when   XylemeCompetitors.ChangeInMyProducts
```

This requests the system to reevaluate the query whenever it detects a change in the *products.xml* page.

4.6.3 Reporting

The report part of a subscription has the following form:

```
select    ...          % report query
when      ...          % reporting condition
(atmost)  ...          % limiting conditions
(archive) ...          % archiving information
```

The report query is a standard Xyleme query that takes as input the current set of notifications, i.e., an XML document, and produces another XML document. The *when* clause tells when to fill in a new report. It follows the syntax:

$$
\begin{array}{lcl}
\langle WhenClause\rangle & := & immediate \mid \\
 & & \langle Event\rangle \ (or \ \langle Event\rangle)* \\
\langle Event\rangle & := & \langle frequency\rangle \mid each \ \langle date\text{-}pattern\rangle \mid \\
 & & \langle count\rangle \ > \ integer \mid \\
 & & \langle notification\_name\rangle \\
\langle frequency\rangle & := & dayly \mid weekly \mid biweekly \mid monthly \\
\langle count\rangle & := & count \mid \\
 & & count \ "(" \ \langle notification\_name\rangle \ ")" \\
\langle date\text{-}pattern\rangle & := & monday \mid tuesday... \mid \\
 & & January1st \mid January2nd...
\end{array}
$$

where $notification\_name$ is the name of a monitoring query (e.g., UpdatedPage). The semantics of these report conditions with a few exceptions should be clear. A condition of the form $count > 500$ means that a report is generated whenever 500 notifications have arrived. A condition $count(UpdatedPage) > 10$ means that the report is generated when 10 *UpdatedPage* notifications have arrived. Immediate means that as soon as something is added to this subscription, a report is generated. The disjunction of several conditions means that a report is generated whenever one of the reporting conditions holds. The generation of a report for a subscription empties the global buffer of notification answers.

The *when* clause is compulsory whereas the last two clauses are optional. The *atmost* clause sets a limit to the reporting query. For instance, *atmost 500* means that after 500 notifications, we will stop registering the new notifications until the next report. Also, *atmost weekly* means that, we do not send a report more frequently than once a week even if the *when* condition "triggers" more often.

Finally the *archive* clause requests the results of this particular subscription to be archived for some period of time: For instance, *archive monthly*

requests to archive the reports for this particular subscription for a month before garbage collecting them. We refer to [84] for more details on the *reporter*.

4.6.4 Controlling subscriptions

It should be noted that the cost of some monitoring or continuous queries may be quite prohibitive. This is the reason why we only allow the condition *extend URL*, and not the matching of an arbitrary pattern. Similarly, one would like to prevent the use of *contains* conditions on too common a word such as "*the*" or attempting to refresh every day all documents in too wide a domain such as *biology*. As a last example, we do not want to trigger a continuous query with too frequent an event, e.g., an event that would occur every minute. To control this, we could use a cost model to estimate *a priori* the cost of a subscription and to restrict the right of specifying expensive subscriptions to users with appropriate privileges. Perhaps a simpler solution would be to allow arbitrary subscriptions, but inhibit them *a posteriori*, if the system finds out they require too much resources.

To the same end of avoiding the waste of resources, we mentioned in the introduction the possibility of using techniques as in [78, 41] to factorize the work in monitoring and continuous queries. Although we have not introduced such optimizations yet, we do however let some subscriptions share monitoring and continuous queries from other subscriptions. These are called *virtual subscriptions*. For instance, a user may request the following subscription (purely virtual):

```
subscription MyVirtualXyleme
virtual MyXyleme.Member
```

A user specifying such a subscription is simply registering to a subscription owned by another user. In other words, we distinguish here between the possibility of creating monitoring and continuous queries (expensive for Xyleme) with that of subscribing to them (that only puts stress on the *Reporter*).

4.7 Alerters

Alerters are the first step of the notification chain. When a document is being retrieved, it is first handled by the URL manager. If it is an XML document, it is managed by the XML loader. (For HTML documents, the story is a bit different but similar and will not be considered in detail here.) Alerters are in charge of detecting atomic events and sending them to the

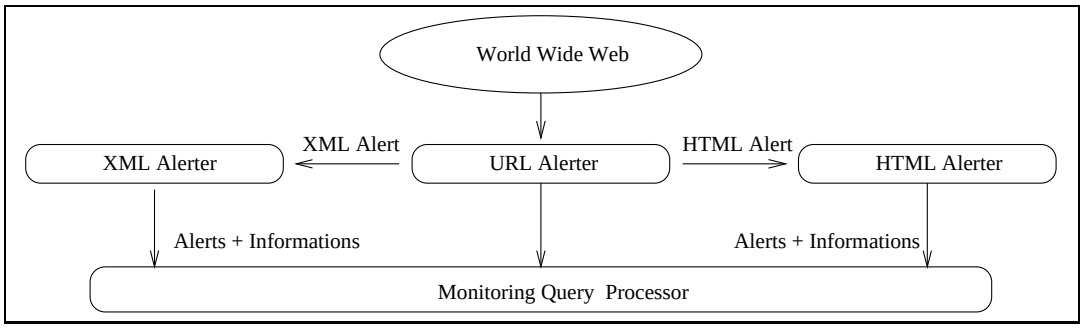


Figure 4.9: Overview of Alerters architecture

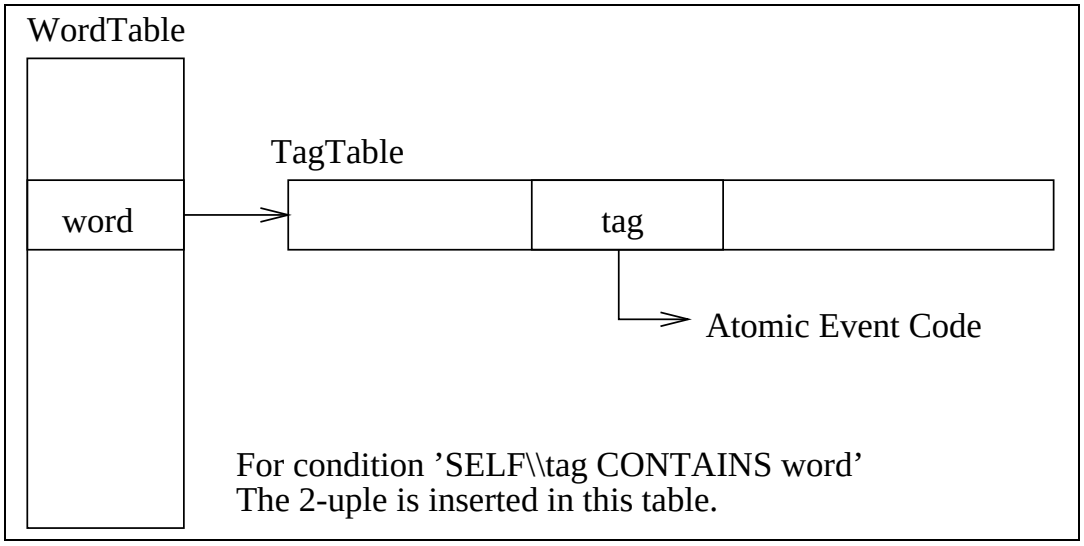


Figure 4.10: Registering (Tag, Word) conditions

Monitoring Query Processor. We next consider the architecture of alerters, then discuss two particular kinds of alerters, the URL and the XML Alerters.

4.7.1 Architecture

To avoid unnecessary network traffic, alerters have to be set as close as possible to the modules they monitor. For instance, the URL Alerter must be placed next to the URL manager (that is gathering meta-data about documents), and the XML Alerter next to the XML Loader (that is in charge of loading XML documents). An *Alerter* plugged into a specific module first of all interacts with its host to get the information it needs to detect the alerts. Then it communicates to other modules (see Figure 4.9):

1. at the end of the chain, the *Alerter* sends its alerts and their associated information to the *Monitoring Query Processor* that is in charge of handling them.

2. an *Alerter* typically sends the atomic events it detected on a document to the next *Alerter* that will further process this data.

For performance reasons, each alerter uses different threads for input and output.

An essential aspect of this process is that we collect all the atomic events of interest on a given document before sending them to the *Monitoring Query Processor*, and thus the Monitoring Query Processor receives simultaneously all the atomic events concerning a document. For instance, for an XML document, some atomic events may be detected by the URL Alerter, then sent to the XML alerter that may detects more atomic events. A single alert is then sent to the *Monitoring Query Processor*. Based on the content of these alerts, the system may decide to do more processing, e.g., send a notification to the Reporter or to the Trigger Engine. Thus we use an approach that is typically based on a *document flow*, vs. work flow.

4.7.2 URL Alerters and pattern detection

In this section, we briefly consider the URL Alerter and, in particular, its main task, namely, the detection of patterns in URLs.

We focus on the detection of URL patterns that is by far the most critical in terms of performance. We handle three kinds of URL pattern detections:

1. URL extends string (e.g. www.xyleme.com/*)

2. filename = string (e.g. */Xyleme2000.xml).

3. URL = string (e.g. www.xyleme.com/index.html)

Each one of them is handled using a separate data structure. To handle *extends*, given the URL of the document that is being fetched, we look up each of its prefixes to see if it matches the 'URL*' pattern of some atomic event of interest. The dominating cost is the look-up in the million-records hash table. To obtain a linear lookup cost, we tried using a dictionary structure. This improved the speed by about 30 percent. But in terms of memory size, the overhead was too high.

The role of the URL Alerter is to construct, for a document, the sequence of atomic events that have been detected on a document. It must produce a sorted sequence since, as we have seen, the Monitoring Query Processor takes advantage of the ordering. We expect such a sequence to be reasonably small (at most a few thousand).

The URL Alerter must be able to manage millions of atomic events, and detect such events on the incoming data (here some meta-data about the page that is being fetched) without slowing down the rest of the system. The main data structure of the URL manager is devoted to this detection. (Again, we will ignore here the issues of the runtime update of the data structure.) To be more precise, we use several data structures depending on the nature of the conditions, e.g. one for *URL extends string* and a different one for *domain = string*. Our data structures are standard and essentially use hash tables and extensible arrays implemented with STL [137].

Observe that the URL Alerter is working concurrently with the URL Manager. Typically, alerters should not slow down the process they monitor by holding up resources. Thus, in particular, it is important that the alerters be able to handle many documents concurrently.

4.7.3 XML Alerter

The main purpose of this alerter is to detect atomic events of the form:

(⟨*change*⟩) ⟨*element-name*⟩ ((*strict*) *contains string*)

For the detection of changes (elements inserted, updated, etc.), we compute the *delta* between the document that is being loaded and its previous version (if available) [98]. We will ignore this aspect here and will focus on the detection of words of interest in the document:

⟨*element-name*⟩ (*strict*) *contains string*

The semantics of *contains* are that the string must be found within the subtree of an element of that name. On the other hand, *strict contains* means

that the element with this tag directly contains this word, more precisely in DOM terms [58], a node with this particular tag has a data child containing this particular word.

Our algorithm relies on the postfix traversal of the DOM tree. For each node n in the tree, let $p(n)$ be the pair: $(level, content)$ where:

1. *level* is the level of the node in the tree;

2. *content* is either the tag if the node is an element node or the data if the node is a data node.

Let $\{n_i\} = \{n_0, n_1, n_2, ...\}$ be the list of the nodes in a postorder traversal of the tree. To handle *contains*, we use the flow of $\{p(n_i)\}$. While processing $p(n_i)$ in the flow $\{p(n_i)\}$, it is easy to have the list of words in the subtree rooted at n_i and those directly below n_i. To do that, two different data structures are used (See Figure 4.10):

1. For each "interesting" word w, we use a hash table called $TagTable[w]$. For each tag l such that (l, w) is an atomic event of interest, the table provides its code. The *TagTable*'s can be accessed from another hash table named *WordTable*. Due to the different nature of condition *contains* and *strict contains*, we need two data structures, one for each.

2. Let us first consider *contains*. For the document that is being processed, we use a data structure that provides for the node being processed, the list of words of the tree rooted at this node, and this at no cost. This is where we benefit from the post-ordering in the input to the algorithm. We can essentially handle the words in a stack of lists of words. Note that we can save some space and processing by keeping in this structure only words that are interesting, i.e., are entries in the contains *WordTable*. Now let us consider *strict contains*. It is somewhat similar since two data children of the node may be separated by an element node, so we have to process the trees rooted at child element nodes before processing the node itself.

Observe that this second data structure may contain at some point in the worst case all words of a document, i.e. be roughly of the document's size. With respect to time, we may have to perform one lookup for each word of the document at each level of the document, which leads in the worst case to $Size \times Depth$, where *Size* is the number of words in the documents, and *Depth* is the maximum depth of the tree structure of the document. For XML documents found on the Web, it turns out that the depth of the document is rather small, so on average, this is an acceptable cost.

In our experiments, the *Alerters* could easily support the rate of fetching documents on the Web imposed by the crawlers and URL managers of Xyleme.

4.8 Conclusion

In this chapter, we have shown how to construct the monitoring module of a large scale Web Warehouse, by providing both a declarative specification language, and an architecture. We conducted a formal study, and experimentations that illustrate the behavior of our algorithm on large quantities of data. The results prove that this system is able to withstand a load of over a million documents processed per day, which we believe is a sufficient order of magnitude.

This system is currently used in production by Xyleme. A noteworthy application is the use of the monitoring system in conjunction with the crawling mechanism of the *e.dot* project: the atomic events are made of the detection of specific keywords in the documents in the flow, and the complex events are conjunctions of two or three atomic events.

We envision improvements to this system more in quality than in performance, by increasing the expressivity of the subscription language to take into account more powerful logic operators, however the current language has proven sufficient for the use with typical focused crawling systems.

Chapter 5

Link Semantics and Thesus

Related Publications: Thesus Web Document Clustering [111, 144, 110, 77]

5.1 Introduction

In many different fields of science, ranging from sociology to physics, there is the common paradigm that an entity is usually *more than the sum of its parts*. However, when applied to searches on the Web, the traditional process is to consider that a page is defined exclusively by its content. On the one hand, World Wide Web surfers typically submit queries to search engines by using one or more keywords to define their interests. On the other hand, results are equally hard to tally, since search engines usually return huge lists of URLs many of which can be judged almost irrelevant to the query.

In classical search engines (Google, Yahoo, Alta-Vista), a Web pages semantics is derived from the words that are assigned to the page, usually because they appear in the body of the page. Yet the World Wide Web is a graph. More precisely still, it is a directed labeled graph, where the pages represent the nodes, and links in the pages represent the edges of the graph. In this chapter, we will show how to add semantic labels to the edges of the graph of the Web, and query the results. We believe that we will be able to extract crucial semantic material about a Web page from the pointers to that page. This is what differentiates the World Wide Web from a simple collection of documents. Links are often cornerstones in Web design, and as such deserve to be taken into account when answering queries. We want to group Web pages into thematic subsets of World Wide Web documents called Thesus (Thematic Subsets of the World Wide Web). The semantic proximity of the pages in a Thesu is not only derived by the pages content

but also by the semantics of the links pointing to this page (link semantics).

The thesis developed in this section is as follows: *Link semantics enrich the semantic content of World Wide Web pages and can be of great value when searching the World Wide Web for retrieval of information.*

The contributions in this chapter are the following :

- a model and a language that enable thematic selection of World Wide Web subsets and subsequent information enrichment by extracting descriptions from the links pointing to the pages of the subset, and also queries that are based on connectivity and semantic information generated by mapping extracted keywords to an ontology. We show that querying links yields exploitable results.

- a mechanism that extracts keywords and enhances hyperlinks with semantics, by mapping sets of keywords that describe a Web page, to sets of concepts (categories) organized in a hierarchy. In the current implementation the mechanism employs WordNet[159] as a thesaurus and an ontology based on DMOZ[117] , the Wu and Palmer similarity measure[160] for computing similarities between terms in a hierarchy. A detailed description of this mechanism is given in Section 5.3.

- a similarity measure for weighted sets of terms in a hierarchy to evaluate the similarity between Web documents. Both documents and users queries in Thesus are seen as weighted sets of terms. The similarity between sets is not based on the exact matching of terms in the two sets but on the combined similarity between terms in the first set and terms of the second set. As a result, queries may retrieve documents that do not contain a certain keyword but a synonym, hypernym or hyponym of it, and that are nonetheless very relevant.

- a clustering mechanism that employs the aforementioned measure to group similar pages together. Two distance (similarity) based clustering algorithms have been adapted for our purpose, to perform Web document clustering.

- a fully implemented client/server system, called Thesus. The server a. collects URLs and pages contents based on a given set of keywords, b. extracts keywords from the collections incoming and outgoing links (by processing links neighboring text in the source URL), c. maps keywords to categories in the ontology, d. clusters pages based either on the keywords or on the categories assigned to them, e. populates a relational database with the results.

- a set of additional services that characterize Web documents or sets of documents using their incoming or outgoing link semantics. These are Web services that allow users:
  - to provide a set of URLs and get the terms that frequently appear in their incoming or outgoing hyperlinks
  - to provide two sets of URLs (A and B) and get the terms that frequently appear in the hyperlinks from URLs in A to URLs in B.

These services are illustrated at the Web site `http://www.db-net.aueb.gr/thesus/` and demonstrate some of the simple services that can be created with our link manipulation language.

My more specific contributions regard the mapping of keywords, the similarity measure and its application to clustering, and the subsequent implementation and experimentation, in conjunction with Iraklis and Maria. This chapter describes the Thesus language, as proposed by the Thesus work group, and takes a closer look at the modules I specifically worked on : the enhancement and clustering modules.

5.2 Motivations

Motivation for link processing:

As already mentioned the World Wide Web is a directed labeled graph whose edges carry information from the source node to the target node. Though link analysis and link structures are exploited in order to rank the importance of pages [70, 83], the links semantics are generally ignored by search engines. Only Google allows a user to query the (exact) text contained in hyperlinks. We believe that crucial semantic material about a Web page can be extracted from the pages that point to that page. In this chapter, we will show how to add semantic labels to the edges of the graph of the Web, and use them for both characterizing the target nodes contents and clustering nodes into related groups.

Often, the information in a page is not self contained. Meta-information absent from the page is useful when answering queries such as looking for a car advertisement, looking for a book, or looking for a picture. While what a page talks about can be derived by what is written in the Web page, there is some added value to also take into account what others think about the page. This is very useful in the case of multimedia content, such as pictures, sound or video files, whose content cannot be otherwise characterized. This

extra information can be manifold. It can be for instance a judgment on the quality of the Web page, such as a link that would read :

```
<a HREF="xxx">a GREAT link on pop bands!</A>
```

It could also be information on the kind of document that would be returned:

```
the <A HREF="xxx"> slides</A> of my presentation
```

Let us note that the interesting information is also not always limited to the contents on the $< A\ HREF >\ < /A >$ tag. One only needs to randomly browse a few Web pages to see that there is a lot of information to be retrieved for the neighborhood of the links. Indeed we will consider the text located around links, and not only the text that form the links themselves, contrary to what previous systems do, like Google.

Assume a document U is being pointed to by a set of incoming hyperlinks $\{(l_i, s_i)\}$ where s_i are the semantics of link l_i. We claim that our knowledge of U is affected significantly by the semantics of the incoming links. Let us consider the following example: If a node U is pointed to by a link, emanating from node S, bearing the semantics *databases*[1] there is a strong indication that the source document S points to the target document U characterizing it with the term *databases*. If there is more than one link from different nodes that point to U bearing the same semantics (*databases*), the indication (as a collective consensus) that U is closely related to the area of *databases* is stronger, and moreover with high importance.

Some of the incoming links of a page are the navigation links that usually provide abundant and possibly incorrect information for the targeted page (i.e. "back", "next", "home" etc.). Additionally, intra-site hyperlinks (links that point to pages in the same host), may carry misleading information on purpose, and attempt to affect Web-agents that extract information from hyperlinks.

A common solution to these problems is to ignore all the intra-site links when performing link analysis. One could also argue that it is better to disregard intra-site links, since they could also provide biased information about pages. However, in our system, information provided by such links does not necessarily affects the final characterization of a Web page. Keywords that are commonly used for navigation purposes (i.e. back, next etc) can be included in the list of stop-words and consequently ignored by the keyword extraction module. Furthermore, keywords that are not included in the stop-words list but are irrelevant to the domain of interest (as it is expressed by the ontology) will not be mapped to a node in the ontology, while keeping links that are relevant.

[1] We will show in this chapter how to characterize the semantics of a Web page

The use of ontologies:

Web documents are mainly characterized by extracted keywords and by a rank that takes into account link structures [30]. Computing a similarity between documents based on exact matching between these terms is not satisfactory. For instance, a document d_1 characterized by the keyword list: $d_1 = \{snake, desert\}$ would be judged irrelevant to a document d_2 characterized by the list: $d_2 = \{adder, Sahara\}$ yet it is arguable that the two lists of keywords (and thus the documents) are in fact related, since an *adder* is a *snake* and the *Sahara* is a *desert*, therefore d_2 deals with the same concepts as d_1, they are just more specialized. By replacing keywords with concepts and moreover concepts in a hierarchy, a more flexible document matching process than strict matching can be achieved, handling both specializations and generalizations of terms.

The system currently uses a simplified form of "ontology". The terms are connected using only one type of relation (a term X "is subclass of" another term Y, which denotes that X is less general than Y). Concepts form a directed acyclic graph that resemble a taxonomy.

We understand "ontology" as a human attempt to organize knowledge in a domain by describing concepts as nodes (with several properties) of a graph and the relations between concepts as directed edges of this graph. More generally, relations between concepts may represent containment (has-a relation), generalization-specialization (is-a), or any other kind of relation that a human may define, thus creating a multicolored directed graph. In this scope, any organization of concepts with at least one type of relation between them can be considered as "ontology". As a result "taxonomy" is itself an ontology, although it is very simple. Note that in the following, in order to apply the results using our similarity measure, our ontology is simply considered to be a tree, but the similarity measure could easily be extended to a Direct Acyclic Graph.

The idea of replacing extracted keywords with semantics from an ontology does not aim to identify the complete - real semantics of hyperlinks. The main gain is that a large set of extracted keywords, which may not be of interest to the user, is mapped to a narrower set of concepts, which are organized in a hierarchical structure and represent the users domain of interest. Users queries are also represented as keyword sets and can thus be mapped to concepts in a similar way. As a result we reduce the dimensionality of the problem of matching documents to queries and moreover we perform approximate matching based on concepts instead of *exact keyword matching.*

5.2.1 Thesus:

We believe that all the above provide a strong motivation for a language and system that enables the definition and manipulation of thematic subsets of the World Wide Web with rich semantics. We will present a system to create collections of thematically relevant pages from the World Wide Web and to further distill them into smaller subsets based mainly on their semantic similarity and connectivity features. The salient features of Thesus are:

- creation of Web document collections on a specific domain of interest, based on a small set of keywords,
- automatic characterization of sets of Web documents in two stages: a) keyword extraction from the incoming links to the sets, b) computation of the semantics these keywords convey, using an ontology and a thesaurus.
- reduced information storage, for a Web warehouse, since for each Web document only the incoming link semantics are stored instead of the whole text,
- organization of document collections into subsets, each of which contains documents with similar semantics or similar connectivity features.
- efficient searching on the created collections, that focuses on the subsets that match users queries.

The system has been implemented in Java. We present its architecture, and illustrate with examples the services it offers, some of which are quite novel.

NOTE: Thesus is the name of the system. A Thesu is a thematic subset, and the plural is Thesus. We refer indifferently to a document, a page or as we will see later, a node, in order to define the basic entities stored in our system.

Thesus is a system that enhances the semantic organization of a set of pages and guides the user within this set. It is envisioned as a personal service that assists the user in the creation and querying of rather compact high quality thematic collections of pages. To give a real feeling of what sort of results can be achieved by the system, we compare answering queries in Thesus to some popular search engines.

5.3 Thesus MODEL

5.3.1 The model in brief

In this section, we provide an informal introduction of the Thesus information model that motivates the formal specifications that will follow. Let us begin with the formal definition of the basic data types. We assume that the World Wide Web is a collection of:

- *pages* uniquely defined by their URL. Each page has a content, a piece of text. Henceforth, we will interchangeably use the terms page, node and document for representing World Wide Web pages. Let us underline that pages can be empty of content (for instance in the case of a picture) yet have other pages pointing to it, with meaningful semantics.
- *links* which connect pages. A link is uniquely defined by its source and target nodes. In our system, links between two pages are perceived as bearers of semantics, thus we are interested in the union of the semantics of all the links pointing from a given page to another given page, to be defined next. The exact location of the source or target anchor (i.e. the specific location in the source page from which the link emanates) is not of interest.

Link semantics: Assume two pages the source page S and the target page T and the set of links $\{l_i\}$ that emanate from S and point to T. Also assume a procedure that for each link l_i returns a list of keywords $\{k_j\}$ that characterizes the link. When authors of a page want to create a link to another page, they use a small set of keywords to describe the target page. These keywords either appear in the hyperlink source (the text that acts as hyperlink), or in a short area around the hyperlink. In the case that an image is used as the hyperlink source, authors usually use the *alt* attribute to describe the target document. We call this information *link keywords*. Given an ontology and using WordNet, we are able to map extracted keywords to concepts of this ontology. The semantics we attach to links is a set of concepts therefore it is important that we convert link keywords to what we call **link semantics**.

The set of keywords (and as we will see further, the set of concepts they correspond to) that can be derived from $\{l_i, \{k_j\}\}$ semantically define the classification of page T seen from page S, in other words the union of the sets of keywords $\{k_j\}$are the **keyword** properties that S assigns to T.

An extension would be to consider a set of target pages $\{T_i\}$ instead of just a single one. Then the result of processing the links from $\{S_i\}$ to $\{T_i\}$

will be the collective characterization that $\{S_i\}$ assign to $\{T_i\}$. This justifies the definition of an operator, `groupKeywords`, that takes as input a set of source pages $\{S_i\}$ and a set of target pages $\{T_i\}$ and returns a list of keywords that result from processing all the links between any page in $\{S_i\}$ to any page in $\{T_i\}$.

Need for crawling In order to retrieve *link keywords* it is necessary to traverse the World Wide Web. We need to find the pages that are being pointed to by the outgoing links for a page S and to find the source pages of a pages incoming links. This crawling function can be carried out in several levels in the World Wide Web graph starting from some node under consideration.

5.3.2 Thesus Model data types

In this section we precisely define the entities that constitute the nucleus of the Thesus language design. Our reference space is the World Wide Web that is perceived as a collection of documents (`w-docs`) connected with links (`w-links`). For a thematic area we assume a subset of the documents and links of the World Wide Web (`docs` and `links` respectively) form a thematic subset (Thesu). For completeness, we assume the data types `URL, keyword` and `ontology-term`. As `URL`, we define the unique identifier of a page in the `World Wide Web` context and as `keyword` a string of characters characterizing a page. We also consider sets of `URL, keyword` and `ontology-term` as types in our system. We will now define the fundamental entities necessary for Thesus creation and manipulation.

Thesus Model entity definitions

Definition 5.3.1 *A collection of documents, `docs` is a set of tuples of the form (URL, {`keyword`}, {`ontology-term`}), where `URL` is the identifier of the document as it appears on the World Wide Web, {`keyword`} is a set of keywords characterizing the document, {`ontology-term`} is the set of corresponding terms of the ontology that semantically describe the document, and have been constructed from the set of keywords.*

This is an extensible definition. It is of course possible to add extra information, which would be stored in an `other-info` field, to represent such information as the content of the page perceived as a string of characters (text) or the last date of change of the document (modification date). Let us note that {`keyword`} is extracted using the link operators, and that `other`

`info` is constructed at a later date. The documents in such a collection constitute a Thesu.

Definition 5.3.2 *A collection of links,* ***links****, is a set of vectors of the form* ***(URLS, URLT,*** $\{$***keyword***$\}$***,*** $\{$***ontology-term***$\}$ ***)****where* ***URLS*** *is the URL of the document from which the link emanates,* ***URLT*** *is the URL of the document to which the link points,* $\{$***keyword***$\}$ *is a set of keywords characterizing the document and* $\{$***ontology-term***$\}$ *is a set of terms from the ontology that semantically define the document.*

World Wide Web Entities

In order to construct `docs` we need to extract information from pages of the Web. The following collections, `w-docs` and `w-links` are used during the construction. In a nutshell, these entities are our vision of the World Wide Web as a virtual entity. We are able to materialize portions of it by using Web services, such as search engines.

Definition 5.3.3 *The World Wide Web collection of documents,* ***w-docs****, is a set of vectors of the form* ***(URL,text)****, where* ***URL*** *is the identifier of the document as it appears on the World Wide Web,* ***text*** *is content perceived as a string of characters.*

Definition 5.3.4 *The links in the World Wide Web,* ***w-links****, is a set of vectors of the form* ***(URLS, URLT)****, where* ***URLS*** *is the URL of the document from which the link emanates,* ***URLT*** *is the URL of the document to which link points. Here we assume that all the links between two pages are aggregated in one vector in the w-links entity.*

5.4 Thesus Language

5.4.1 The basics

In this section we introduce Thesus Language that enables thematic selection of World Wide Web subsets and subsequent enrichment by extracting semantics from the links pointing to the pages of the subset. The Thesus language also provides means to submit queries that are based on connectivity features and related semantics, yielding meaningful results that are otherwise not obtainable from existing search mechanisms. We formally define the fundamental operations that are necessary for defining a Thesu.

When designing a language, the issue of *minimality vs. expressiveness* usually arises as a trade off. On the one hand (*minimality*), we are searching for a minimal set of operators that minimizes the overlap of semantics between them, are simple in their design and by composing them construct richer operations. On the other hand, there is the requirement for ease of use, in the sense that the set of operators should be easily understandable by the potential users and with clear and close to the applications semantics.

In the Thesus model, we made a compromise between these two aspects by providing a set of operators clearly grouped in distinct categories (**crawling**, **links semantics** extraction, **link analysis**) aimed at different tasks. At the same time the operators are semantically rich in the sense that they are directly applicable to the domain considered (i.e., traversal of the World Wide Web graph and semantics extraction). Nonetheless, there is some overlap between some of the operators and in particular certain operators can be expressed in terms of the others.

Another important issue is closure, i.e., the fact that any simple or complex operator returns a type that is already defined in our system. Thesus type system is closed under the specific set of operators since all operators introduced in this section return values or sets of values with in the data type system we introduced: sets of URL or sets of keyword, and relations over these basic domains. Let us note that the elements of type `ontology-term` are used as a specialization of the `keyword` type in some of crawling operators defined later on. The transformation from one to the other is done by the operator `getSemantics({keyword})`.

Apart from the core set of fundamental operators, we considered defining specific operators that are useful in order to further process Web pages. Potential users of the system can easily define other operators in terms of the fundamental ones, if these operators limit themselves to the use of the basic predefined types as given in 5.3.2. We give here the example of operators that deal with other features in link analysis. Such features relate to aggregate connectivity features and convey important information on a set of pages.

We consider the following features:

- number of outgoing links (a page is considered as a *hub* [83] if it has many outgoing links);
- the number of incoming links (here we have the *authority* concept [83] where a page is being pointed to by many links);
- link patterns such as co-citations [133] and couplings [87]. The last two features are considered as indications of similarity between pages. More specifically, co-citations measure the number if out-links in common

between a set of documents whereas coupling measures the number of in-links that cite all documents in a set of consideration [153].

Kleinberg in [83] provides a more complex definition of hubs and authorities. According to his definition, each page has a hub and an authority score, which is calculated recursively: hub score increases if the page points to many pages with high authority scores and inversely authority score increases if the page is pointed by many good hubs.

In Thesus for simplicity we use the one step definition. We exploit and enrich the above concepts with semantics extracted from the links. We introduce the following concepts:

- *Thematic Hubs:* As mentioned before, a hub is a page with many outgoing links. Some of the outgoing links can be of similar semantics. This is an indication that such a document is important to visit as the user will find lots of links to other documents. We capitalize on this concept and we enhance the semantics of a hub by considering the semantics of the outgoing links. It is obvious that a hub with many outgoing links of similar semantics is a thematically focused one and potentially a more interesting one.

- *Thematic Authorities:* An authority is defined as a document with many incoming links [83]. We enhance the semantics of an authority by considering the semantics of the incoming links. For instance if an authority node is pointed to by links of type databases it is very likely that this is a page that contains reference material on databases. Also the co-citation concept can be exploited for further enhancement of the semantics of a node. If there are links from nodes B and C towards node then A is said to be co-cited by B and C. Moreover if the link semantics are similar then As semantics are further focused and also we have an indication that B and C have some similarity.

- *Page semantic description:* As discussed earlier, a document's classification should be highly affected by the semantics of the incoming links. Therefore by taking into account in a collective manner the semantics of all the incoming links it should be possible to devise the semantic description of the document. Let us not forget the coupling concept that enables similarity assignment between nodes if they are pointed to by links emanating from the same nodes bearing similar semantics. For instance if documents A and B are pointed to by C with links of similar semantics then there is an indication that A and B are of similar content (coupling)

In the next section we build on the set of basic operators and we define additional operators conveying richer knowledge about a set of World Wide Web pages in the lines described above. This will illustrate the potential of Thesu operators in terms of finding rich semantics in the World Wide Web.

5.4.2 Crawling operators

We introduce four crawling operations:

Definition 5.4.1 ***fetch(URL)*** *Given a* `URL`*, it returns either its textual content or null in case the link is broken.*

The next operator takes as an argument a URL expression and returns a set of URLs that match the expression. While we do not intend to go into any details in the construction of a crawler, we demand that the system used complies to the following specifications.

Definition 5.4.2 *Given a set of URL expressions,* {`URLexpr`} *the operator* ***groupMatch({URLexpr})*** *returns a set of URLs that match the expression* `URLexpr`*. The expression is a partial URL string and all returned URLs should begin with the URL expression.*

Example For instance the operation `groupMatch({http://www.nasa.gov/, http://www.ibm.com/})` returns all the pages under http://www.nasa.gov/ or http://www.ibm.com/

Definition 5.4.3 *Given a set of URLs ({*`URL`*}), and an integer N the operator* ***crawl({URL},N)*** *returns a set of URLs, consisting of the pages that are being pointed to from each page in {URL} in each level of the World Wide Web graph until level N. Positive values of N imply that we crawl the World Wide Web in a forward manner by following the outgoing links of pages in {URL}, whereas negative values of N denote backward crawling through the incoming links of pages in {URL}.*

The example in Figure 5.4.2, shows that for N=1 the crawl operator returns the documents in {URL} and all the documents pointed by them, whereas for N=-1 it returns the documents in {URL} and all the documents that point to them.

Example: In the case of `crawl({`U_1`}, 1)`, starting from page U_1 the operator returns all the URLs that are targeted by links starting from U_1 that are reachable after at most 1 hops.

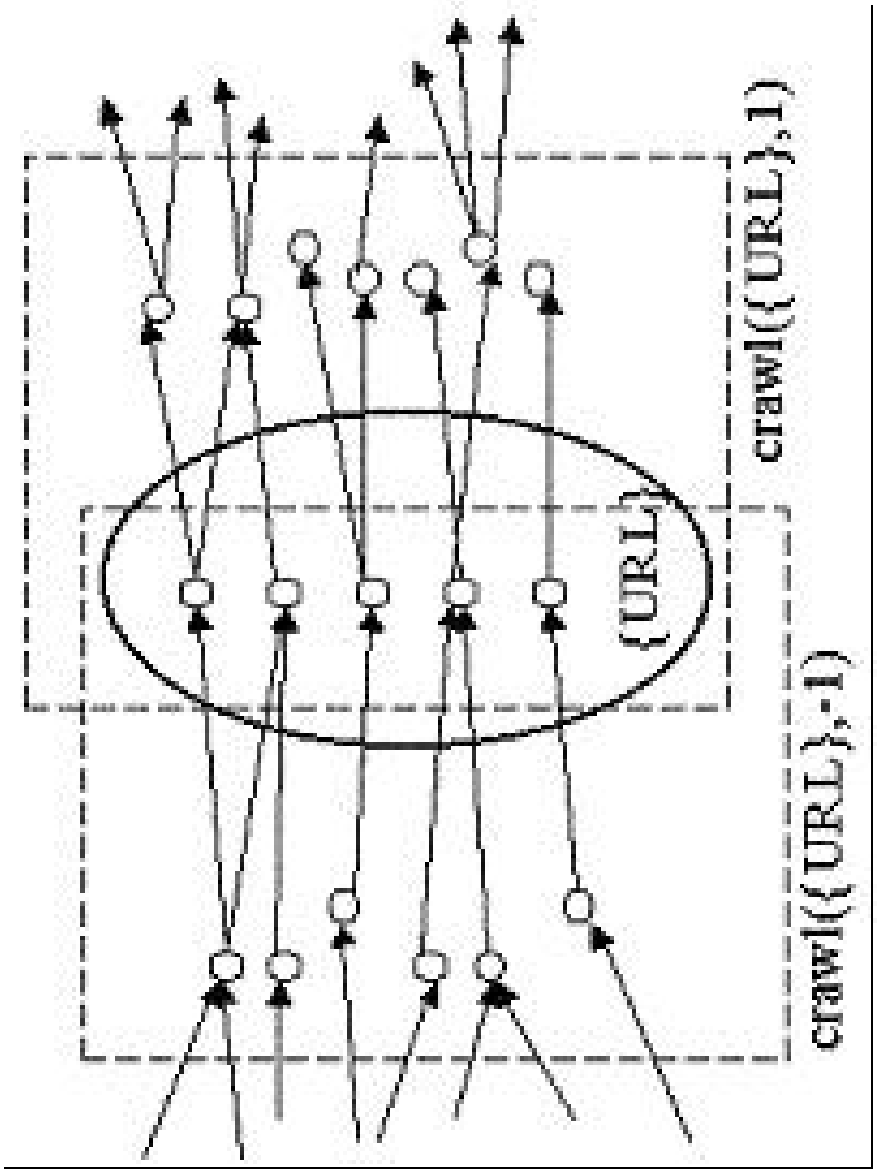


Figure 5.1: Crawl Operator

5.4.3 Link Semantics operators

Here we propose a set of auxiliary functions to be used in the definition of this category of operators. In order to extract the semantics of links we need to be able to locate the position of the source of the link (i.e. $< A\ HREF = >$) in a page and subsequently process its neighborhood in order to extract keywords.

Therefore given the text of a page, `text`, and the string corresponding to a links target `URL, URLT`, the function `getpos(text, URLT)` is used that returns the set of positions {pos} of the string URLTs occurrences in the source node. Then the text neighborhood (i.e. 100 characters) of the link is processed. This is done by calling function `process(w-docs.TEXT, pos, 100)` for each occurrence of URLT. This function returns the set of keywords extracted and represents the keyword semantics of the specific link. In order to map these keywords to categories of the ontology, and thus really access more global semantics, we use the function `getSemantics({keyword})` that returns {`ontology-term`}, and refer to Section **??** for details on how it works. We would just like to stress here that our language manipulates two different entities in order to provide a description of the page: `keywords`, most often extracted from the incoming links of a page, and `ontology-terms`, which represent a more abstract definition of the contents of the page, expressed in the terms of the ontology.

Since we do not take into account multiple links from a page S to a page T, we define the operator `linkKeywords` that returns the union of all keywords that appear in the links between a source page S and a target page T.

In the following definitions, details on implementation are omitted.

Definition 5.4.4 *Given two pages identified by URLs URLS and URLT the operator* ***linkKeywords (URLS, URLT)*** *returns the set of keywords that are extracted from all the links emanating from URLS targeting to URLT.*

Definition 5.4.5 *Given two sets of URLs {URLS} and {URLT} the operator* ***groupKeywords ({URLS}, {URLT})*** *returns the keywords extracted from all the links that point from any page in {URLS} to any page in {URLT}.*

The sets of urls {URLS} or {URLT} in groupKeywords may contain a single url, or can be empty set. In the former case, instead of {URLS} and {URLT} we may have sourceURL and targetURL respectively, so `groupKeywords` functionality reduces to the one of `linkKeywords` operator. In the latter case, when {URLS} is empty `groupKeywords` examines every incoming link

to pages in {URLT}, when {URLT} is empty `groupKeywords` examines every outgoing link of pages in {URLS}. We illustrate the meanings of the operator `groupKeywords` for different combinations of the input parameters with the following examples .

- `groupKeywords({URLS},e)` returns the union of `linkKeywords` between each page d in {URLS} and e.
- `groupKeywords({URLS},∅)` returns the union of `linkKeywords` between each page d in {URLS} and each page d in `crawl({URLS},1)`. This information indicates how pages in {URLS} describe the pages they point to, in other words, how pages in {URLS} perceive the world.
- `groupKeywords (∅, {URLT})` returns the union of `linkKeywords` between each page d that points to any page e in {URLT} and e. This information indicates how their referrer describe pages in {URLT}, in other terms what the world thinks for pages in {URLT}.

Definition 5.4.6 *Definitions 5.4.3 and 5.4.4 are combined to the definition of the operator* ***thematicCrawl({URL},{keyword},N)****, which given a set of URLs ({URL}), a set of keywords ({keyword}) and a integer N returns a set of URLs. This set of URLs consists of the pages that are being pointed to from each page in {URL} in each level of the World Wide Web graph until level N, having at least one keyword from the {keyword} set in the link keywords.*

The operator is called `thematicCrawl` since when it is used with keywords that fall in the same thematic area then it collects URLs that have a high probability to be on similar subjects. Note that we have not yet applied any semantic processing. Here follow the exact semantics in pseudo-code:

```
thematicCrawl({URL}, {keyword}, N):-
if N<=-1
∀ d ∈ {URL}
if (w_links.URLT = d
        AND
linkKeywords(w_link.URLS,d) ∩ {keyword} ≠ ∅)
RES = RES ∪ w_link.URLS
        return thematicCrawl(RES, {keyword}, N+1)
        if N=0
                return {URL}
RES = {}
```

```
if N>=1
∀ d ∈ {URL}
if (w_links.URLS = d
        AND
linkKeywords(d,w_link.URLT) $∩ {keyword} ≠ ∅)
RES = RES ∪ w_link.URLT
         return thematicCrawl(RES, {keyword}, N-1)
```

Note that this algorithm is in no means optimized, it just conveys the semantics of the operator.

Example In the case of `thematicCrawl({`U_1`},{keyword1,keyword2}, 2)`, starting from page U_1 the operator returns all the URLs that are targeted by links starting from U_1, containing either keyword1 or keyword2 and that are reachable after at most 2 hops.

5.4.4 Advanced link semantics operators

The definition of `groupKeywords` operator assumes that the final set of keywords is a union of keywords appeared in each link without counting keyword occurrences. However if we take into account keyword occurrences, the modified `groupKeywords` definition conveys three different meanings based on the way keywords are aggregated: a) if we aggregate per keyword then the output of `groupKeywords` will be a set of pairs (KEY,TIMES) where TIMES represents the number of KEY occurrences in the links from {URLS} to {URLT} (see definition 5.4.7), b) if we aggregate per target page, then TIMES represents the number of target pages that are characterized by keyword KEY (see definition 5.4.8) and c) if we aggregate per source page, TIMES is the number of source pages that use keyword KEY in the links to pages in URLT (see definition 5.4.9).

In the definitions that follow, we use three auxiliary functions:

- `getkeys(`$wkeys$`)`, which takes a set WKEYS = $(key, times)$, where $(key, times)$ is a pair of keywords and number of occurrences, as an input and returns the set of keywords {key},
- `getTimes(`$wkey, k$`)`, which takes a set of $(key, times)$ pairs and a keyword k as an input and returns the number of occurrences ($times$) of keyword K
- `update(`$wkeys, k, n$`)`, which takes a set of $(key, times)$ pairs, a keyword k and an integer n as an input and increases the $times$ value of keyword k by n (updates pair (k,$times$) of $wkeys$ to $(k, times + n)$).

Definition 5.4.7 *Given two sets of URLs* $\{URLS\}$ *and* $\{URLT\}$ *the operator* ***weightedGroupKeywords*** *(*$\{$***URLS***$\}$, $\{$***URLT***$\}$*) returns a set of pairs* $\{$*(KEY, TIMES)*$\}$ *where KEY represents a keyword that appears in the links that point from any page in* $\{URLS\}$ *to any page in* $\{URLT\}$ *and TIMES the number of occurrences of keyword KEY.*

Here follow the semantics in pseudo-code:

```
weightedGroupKeywords ({URLS}, {URLT}) :-
WKEYS = {}
∀ d ∈ {URLS}
      ∀ e ∈ {URLT}
            ∀ K ∈ linkKeywords(d, e)
if getkeys(WKEYS) ∩ {K} ≠ ∅
update(WKEYS,K,1)
       else WKEYS=WKEYS ∪ (K, 1)
return WKEYS
```

Definition 5.4.8 *Given two sets of URLs* $\{URLS\}$ *and* $\{URLT\}$ *the operator* ***weightedTargetKeywords*** *(*$\{$***URLS***$\}$, $\{$***URLT***$\}$*) returns a set of pairs* $\{$*(KEY,TIMES)*$\}$ *where KEY represents a keyword that appears in the links that point from any page in* $\{URLS\}$ *to any page in* $\{URLT\}$*and TIMES the number of target pages characterized by keyword KEY.*

Here follow the operators semantics in pseudo-code:

```
weightedTargetKeywords ({URLS}, {URLT}) :-
WKEYS = {}
       ∀ e ∈ {URLT}
            ∀ K ∈ groupKeywords({URLS}, e)
if getkeys(WKEYS) ∩ {K} ≠ ∅
update(WKEYS,K,1)
            else WKEYS = WKEYS ∪ (K, 1)

return WKEYS
```

Definition 5.4.9 *Given two sets of URLs* $\{URLS\}$ *and* $\{URLT\}$ *the operator* ***weightedSourceKeywords*** *(*$\{$***URLS***$\}$, $\{$***URLT***$\}$*) returns a set of pairs* $\{$*(KEY,TIMES)*$\}$ *where KEY represents a keyword that appears in the links that point from any page in* $\{URLS\}$ *to any page in* $\{URLT\}$*and TIMES the number of source pages that use keyword KEY in their links to pages in* $\{URLT\}$*. Here follow the exact semantics in pseudo-code:*

```
weightedTargetKeywords ({URLS}, {URLT}) :-
WKEYS =
        ∀ d ∈ {URLS}
                ∀ K ∈ groupKeywords(d, {URLT})
if getkeys(WKEYS) ∩ {K} ≠ ∅
update(WKEYS,K,1)
                        else WKEYS=WKEYS ∪(K, 1)

return WKEYS
```

The examples of paragraph 5.6.1 illustrate the different semantics of the various implementations of `groupKeywords`.

5.4.5 Link Analysis operators

The set of Link Analysis operators enhances with semantics the two fundamental operators employed in link analysis -Hubs and Authorities- and two equally important operators *co-citations* and *couplings* that examine the similarity of Web pages based on their common outgoing and incoming links. The definitions are based on a simplified definition of Hubs and Authorities, although the recursive definition presented in [83] can be extended with semantics in a similar way.

Definition 5.4.10 ***Thematic Authority*** *Given a a set of URLs* $\{URL\}$*, an integer* ***threshold*** *and a set of keywords* $\{$***keyword***$\}$ *the operator* ***Tauthorities ($\{$URL$\}$, threshold, $\{$keyword$\}$)*** *returns a set of URLs that have more than* ***threshold*** *incoming links each of which contains all the keywords in the* $\{keyword\}$ *list.*

A simplistic algorithm sketch for the operator follows:

```
Tauthorities({URL}, threshold, {keyword}):-
RES = {}
∀ d ∈ {URL}
        WKEYS = weightedSourceKeywords0,d)
//returns the keywords occurring in the incoming links
//of d and the number of links for each keyword
        if ∀ K ∈ {keyword} getTimes(WKEYS , K) > threshold
//if the incoming links for every keyword in the set
//are more than the threshold
//then the page is a Thematic Authority
RES = RES ∪ d
```

According to the definition a page is considered Thematic Authority if more than `threshold` links point to it each one using all the keywords in {`keyword`}.

Definition 5.4.11 *Thematic Hub: Given a set of URLs {URL}, an integer* ***threshold*** *and a set of keywords {keyword} the operator* ***THubs({URL}, threshold, {keyword})*** *returns a set of URLs that have more than* ***threshold*** *outgoing links each of which contains all the keywords in the {keyword} list.*

A simplistic algorithm sketch for the operator follows:

```
THubs({URL}, threshold, {keyword}):-
RES = {}
∀ d ∈ {URL}
    WKEYS = weightedTargetKeywordsd, 0)
//returns the keywords occurring in the outgoing links
//of d and the number of links for each keyword
        if ∀ K ∈ {keyword} getTimes(WKEYS , K) > threshold
//if the outgoing links for every keyword in the set
//are more than the threshold
//then the page is a Thematic Hub
RES = RES ∪ d
RETURN RES
```

According to the definition a page is considered Thematic Hub if it has more than `threshold` links to other pages each one using all the keywords in {`keyword`}.

Definition 5.4.12 *Thematic Cocitation: Given a set of URLs {URL} and a set of keywords {****keyword****} the operator* ***TCocitations({URL}, {keyword})*** *returns a set of URLs that are all pointed to by every page in {URL} by links whose semantics match the semantics of the list {****keyword****} list.*

The algorithm of the operator is as follows:

```
TCocitations(URL, keyword):-
RES =
N= size(URL)
        //the number of elements in the set URL
TEMP = crawl(URL,1)
        //the URLs pointed by every page in URL
∀ d ∈ TEMP
        WKEYS = weightedSourceKeywords(URL,d)
```

```
        //returns the keywords occurring in the links from pages
        //in URL to d and the number of pages of URL that
        //use each keyword in the links to d.
       if ∀ K ∈ keyword getTimes(WKEYS , K) = N
//if all N pages in URL point to d using K
//and this happens for every K in keyword
RES = RES ∪ d    // then d is a thematic co − citation
RETURN RES
```

According to the definition a page is considered a Thematic Co-citation for a set of URLs {URL} if every page in the set points to this page using all the keywords in {keyword}.

Definition 5.4.13 *Thematic Coupling: Given a set of URLs* $\{URL\}$ *and a set of keywords* $\{$***keyword***$\}$ *the operator* ***TCouplings(*** $\{$***URL***$\}$**,** $\{$***keyword***$\}$***)*** *returns a set of URLs that all point to every page in* $\{URL\}$ *by links whose semantics match the semantics of the list* $\{$***keyword***$\}$ *list. A sketch of the algorithm for the operator follows:*

```
TCouplings(URL, keyword):-
RES =
N= size(URL)
  //the number of elements in the set URL
TEMP = crawl(URL,-1)
  //the URLs that point to at least one page in URL
       ∀ d ∈ TEMP
       WKEYS = weightedTargetKeywords(d,URL)
               //returns the keywords occurring in the links of d to
               //page in URL and the number of pages of URL that
               //are pointed by d using each keyword.
       if ∀ K ∈ keyword getTimes(WKEYS , K) = N
//if d points to all N pages of URL using K
//and this happens for every K in keyword
RES = RES ∪ d
// then d is a thematic coupling
RETURN RES
```

5.5 Semantics management in Thesus

We assume for a specific domain that there exists an ontology O that represents the domain semantics. Each set of keywords $\{k_{d_i}\}$ is mapped to a

set of categories c_j of the ontology O, using a thesaurus (in our experiment WordNet). We use a similarity measure, the Wu and Palmer measure [160] to measure the similarity between one keyword and one category, and we extended it to give a correct assumption of the similarity between sets of keywords and sets of categories. The outcome of this process is that every document in the original set is now enriched with:

- Keywords and weights (indicating the occurrences of a keyword in the incoming links of a page)
- Categories of the ontology into which a document is classified and the respective weights.

We use the following notation to define these enhanced documents:

Definition 5.5.1 *An enhanced document is the triplet (Doc, K, C), where Doc is the document identifier (such as its URL), K (resp.C) is the set of couples $\{(w_i, k_i)\}$ (resp. $\{(v_j, c_j)\}$) of weighted keywords (resp. weighted categories) that define the document (w_i (resp. v_j) is a real, k_i (resp. c_j) is a string).*

Note that w_i and v_j are not necessarily the same if $i = j$. A weight is a real from the interval [0;1]. A value of 1 indicates total relevance, 0 indicates no relevance. A value of 0.2 would indicate low relevance.

Note that these keywords form a positive definition of the document. We do not negatively characterize the document (for instance with words that would not be relevant to the document). This remains for future work.

5.5.1 Semantic enhancement using an Ontology

In the experiment, we used word net, an online lexical reference system, in which English nouns, verbs, adjectives and adverbs are organized into synonym sets, each representing one underlying lexical concept. Different relations link the synonym sets between each other, such as generalization-specialization for verbs and nouns, part-whole for nouns etc. For a verb or noun WordNet provides a *synset*, a set of different senses that the verb or noun may have. Verbs and nouns senses, are organized as topical hierarchies forming a forest of 25 (IS-A) trees of nouns and 15 of verbs. For adjectives and adverbs WordNet provides synonyms.

For each keyword in WordNet we can have a set of senses and in the case of nouns and verbs, a set of generalization path from each sense to the root sense of the hierarchy. To give an example the word wind has 8 senses as verb and 8 senses as noun. For the first sense of wind as a noun we

have the following path: wind → weather, weather condition, atmospheric condition → atmospheric phenomenon → physical phenomenon → natural phenomenon, nature → phenomenon

Ontology In order to abstract extracted keywords to the closest semantics, we need to refer to an ontology of terms that are relevant to our domain of interest. Any hierarchy of terms can be used, provided it can be modeled as a tree. Examples of ontologies that can be employed, are the ontologies suggested by the DARPA Agent Markup Language Program [51](i.e. ontology on music http://www.daml.org/ontologies/276). In order for our system to work, we need to have a mapping of each ontology term to a set of senses (synset) in WordNet. Normally all the senses of an ontology term are considered. However, semantics extraction can be improved if the irrelevant senses are ignored. For example, for an ontology on music, that contains the term wind, it is preferable to select only those senses that relate to music and ignore the senses that relate to weather. This step is optional and requires the user intervention. The system presents all the possible senses given by WordNet and the user de-select those that do not relate to the domain. Following the example on music, for the term wind we keep only 2 out of the 16 suggested paths:

- wind instrument, wind → musical instrument → instrument → device → instrumentality, instrumentation → artifact, artifact → object, physical object → entity, something
- wind, wind up → tighten, fasten → change, alter.

Associating sets of keywords that describe a document to concepts in the ontology The information extraction module generates for each document d_i a set of keywords $\{k_i\}$ with a respective set of weights. In Thesus the weights represent the number of documents that point to d_i using a particular keyword. So each document d_i has the following description: $d_i = (URL, \{(k_j, n_j)\})$ where n_j represents the number of documents that point to d_i using keyword k_j. It is widely accepted that the keyword importance increases proportionally to n_i [30].

Given a set of keywords that characterize a document, an ontology and WordNet, the "semantic enhancement module" attempts to find a set of concepts from the hierarchy that best match this set. For each concept in the hierarchy, WordNet provides all its possible senses (this is called synset in WordNet terms) and for each sense a path starting from an abstract concept (i.e. entity) and specializing down to the specific sense.

In order to find the closest term in our ontology for a keyword k that describes a document, we compute the Wu and Palmer similarity between all senses of $\{k_i\}$ and all senses of all the terms $\{t_i\}$in our ontology. We select the (k, t) pair that gives the maximum Wu and Palmer similarity and map keyword k to the ontology term t.

The first step to improve mapping is to reject the irrelevant senses of the concepts in the ontology. The definition of the senses of each ontology node is crucial for finding the semantics of keywords avoiding wrong mappings, so it is worthwhile doing it manually. In order to further improve mappings, it is essential to reduce the number of senses examined for each keyword k by removing senses that are completely irrelevant in the scope of the ontology. To do this, for each keyword k in $\{k_j\}$ we compute the similarity of each of its senses to the senses of all other keywords in $\{k_j\}$ and keep those that get the highest similarity score. Thus, the weighting set of senses is characterized by high self-correlation.

To provide an example, the self-correlation of the keyword set *(guitar, flute, wind)* gives a score of 0.8 to the triplet of meanings *(guitar, flute/transverse flute, wind instrument/wind)* and less than 0.5 to any other combination. Similarly the self-correlation of the set *(storm, cloud, wind)* gives a score of 0.8 to the triplet of meanings *(storm/violent storm, cloud, wind/air moving)* and lower scores to any other combination. These are indications that *wind* has the sense of *wind instrument* when it appears to the set *(guitar, flute, wind)*, whereas it has the sense of *wind as weather phenomenon* in the set $(storm, cloud, wind)$. As a result the first set will be mapped to the set of categories *(string instrument, wind instrument, wind instrument)* using an ontology on music, whereas the second set will not be mapped to a set of categories related on music.

Based on the remaining senses, each keyword in $\{k_j\}$ is mapped to the closest ontology term as previous. As a result each k_j is mapped to a term t_i with a similarity s_j. It is common that more than one keywords in $\{k_j\}$ are mapped to the same ontology terms, so the cardinality of $\{t_i\}$ is usually smaller than that of $\{k_j\}$. The weight r_i assigned to each term t_i is computed using the following formula:

$$r_i = \frac{\sum n_j \times s_j}{\sum n_j}$$

5.5.2 Semantic Clustering

To be able to run the clustering algorithm, we need a similarity measure over documents. A lot of work has already been conducted in this field ([75]). One

contribution of Thesus is to combine the use of a similarity measure between elements of the set to calculate a more accurate similarity between the sets themselves.

Similarity measure This similarity measure will be used both when clustering the documents, and when answering queries. We expect it will be called upon very often, and thus it is essential that it runs as fast as possible, even on large numbers of documents. Therefore, we need to bear in mind the complexity of the calculation of this similarity. For scalability reasons, it must be independent of the number of documents in the database.

Let us not forget that we are calculating similarity between sets of weighted words, and not simply words. There has been very little research on creating a similarity measure on sets of elements of a space with a similarity measure defined (see [60, 115]). The similarity measure employed in Thesus, called THESIM is a generalization of Wu and Palmers [160] measure and is defined in the following.

Notation 5.5.2 *Let Ω represent the ontology (set of words in a hierarchy).*

Notation 5.5.3 *We use cursive capitals $\mathcal{A}, \mathcal{B}$ to represent sets of weighted words, such as: $\mathcal{A} = \{(w_i, k_i)\}$, and $\mathcal{B} = \{(v_i, h_i)\}$, with $k_i, h_i \in \Omega$ and $w_i, v_i \leq 1$.*

We define :

$$\zeta((A,B) = \frac{1}{2}((\frac{1}{K}\sum_{i=1}^{|\mathcal{A}|}\max_{j\in[1,|\mathcal{B}|]}(\lambda_{i,j} \times S_{WP}(k_i, h_j))) + (\frac{1}{H}\sum_{i=1}^{|\mathcal{B}|}\max_{j\in[1,|\mathcal{A}|]}(\mu_{i,j} \times S_{WP}(h_i, k_j))))$$

with :$\lambda_{i,j} = \frac{w_i + v_j}{2\times\max(w_i,v_j)}$ and $K = \sum_{i=1}^{|\mathcal{A}|}\lambda_{i,x(i)}$ where

$$x(i) = x|\lambda_{i,j} \times S_{WP}(k_i, h_x) = \max_{j\in[1,|\mathcal{B}|]}(\lambda_{i,x} \times S_{WP}(k_i, h_j))$$

Simply put, K is a normalizing factor that is the sum of all the $\lambda_{i,j}$ that were used. In a similar way we can define $\mu_{i,j}$ and H.

Measure intuition : Let us try to briefly explain the intuition behind this measure. This measure basically calculates for each concept in the first set, the concept in the second set it is closest to. It then averages this value, and takes into account the weight of each concept. The weight associated to a concept in reality measures our confidence in the fact that this concept was the correct mapping of the keyword it is associated to. A high value means the mapping is reliable, and a low value means the mapping could be faulty. Thus the weight associated to the matching of a concept in the first document with a concept in the second one is going to be high if the belief that both concepts have a high chance of being related to the document, and lower if this is not the case.

An easier measure to understand is a measure in which the weights associated to all concepts is 1, i.e, if we assume that all the concepts that describe the document are correct. In this case, the measure simply becomes :

$$\zeta(A,B) = \frac{1}{2}((\frac{1}{|A|} sum_{i=1}^{|A|} \max_{j\in[1,|B|]}(S_{WP}(k_i,h_j))) + (\frac{1}{|B|}\sum_{i=1}^{|B|}\max_{j\in[1,|A|]}(S_{WP}(h_i,k_j))))$$

It is in fact this simplified measure that was used in most cases, for performance reasons.

Web document Clustering algorithms The problem is considerably different compared to the case of points in a metric space. In this space there are no coordinates and ordering and the objects to be clustered are sets of (weighted) strings that correspond to concepts of a domain ontology. We only have a similarity measure between sets of weighted concepts. The first algorithm used in Thesus is a modification of the density criteria used in the popular density based algorithm DBSCAN [61]. Modification details and the algorithm itself are described in [77]. An incremental hierarchical algorithm, a variation of the COBWEB algorithm[65], has also been implemented.

The two algorithms operate differently and the final partitioning they produce differs. COBWEB is a hierarchical algorithm that clusters every document in a cluster. The document set difference on the output of the two algorithms is that DBSCAN considers as noise the documents that are different from all other documents. Such documents are not assigned to a cluster, as it happen in COBWEB. COBWEB clusters every document in the same cluster in the first level and in one of its children clusters in the next level aiming to increase the quality of the produced clusters. It merges or splits clusters to this scope. As a result each level in the COBWEB hierarchy contains all the documents in the set but the number of clusters increases

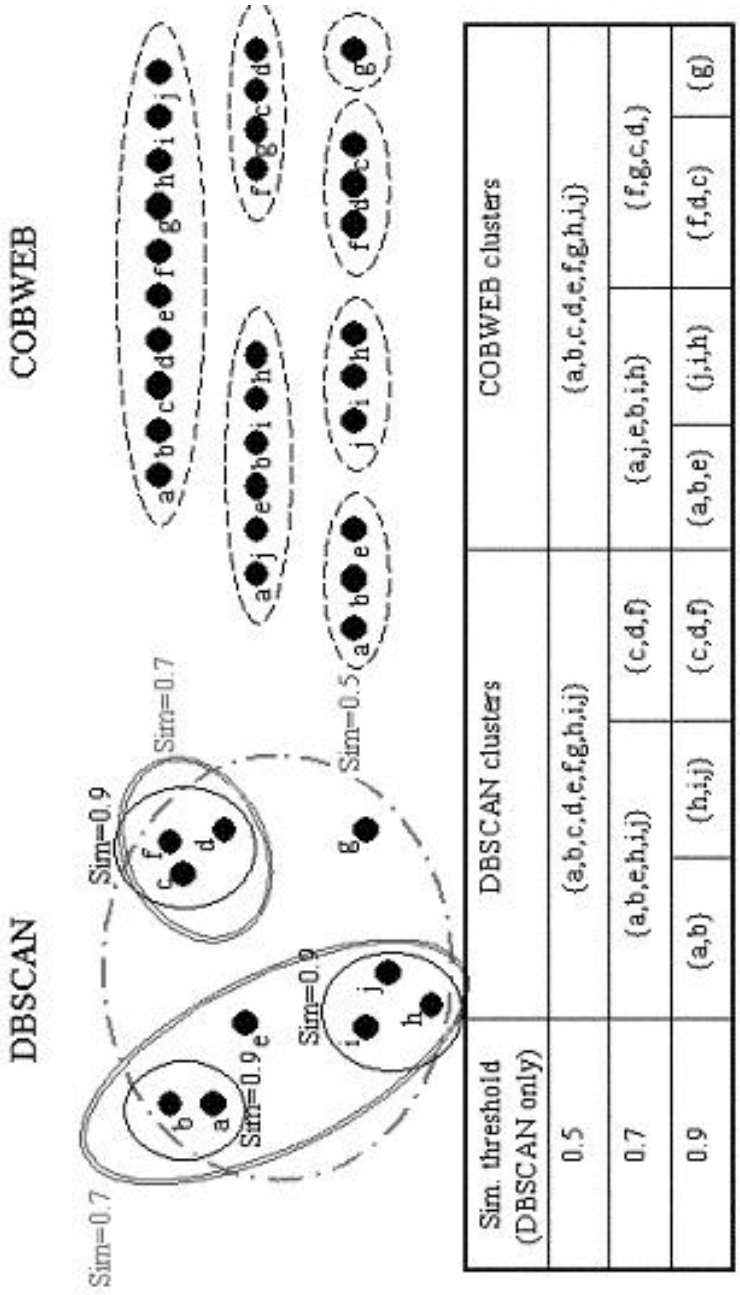
from the top to the bottom levels. On the contrary, DBSCAN produces a flat clustering scheme that does not contain all the documents. By decreasing the similarity threshold (which is a parameter in DBSCAN) for two documents to belong to the same cluster, the size of clusters increases, several clusters are merged and fewer documents are considered as noise. We repeat DBSCAN in the same document set for decreasing values of the similarity threshold and produce a pseudo-hierarchy of clusters. The clustering schemes of the two algorithms are depicted in the figure that follows (Figure 5.2).

Labeling Once the clusters have been found, a very important issue is their labeling (i.e., the assignment of a succinct yet descriptive set of categories to each cluster in order to facilitate user navigation and querying). Grouping documents together is itself a semantic enhancement. We would also like to find appropriate labels for each cluster for the following reasons:

- Simply grouping documents together does not give a way of characterizing this set.
- Cluster labels facilitate browsing through the set of documents as a whole.
- We need some sort of means of computing which cluster a given query is closest to. This would in particular reduce the task of finding on which cluster we should apply a keyword query, to a simple similarity measure between that query (i.e. the set of keywords) and all the labels of the clusters.

The labels are meant to capture the semantics of clusters. The labeling process summarizes in the following:

1. We first of all fix a bound on the number of labels that can characterize a cluster.
2. Construct U, the union of all concepts that appear in at least one document of the cluster.
3. For every concept c_i in U, compute the number of documents in the cluster in which it occurs, or the percentage of documents of the cluster that are characterized by this concept.
4. Reduce the number of concepts in U, using the ontology. In this step, less important categories (with low percentage or very specific ones) are replaced by their superseding categories and then identical categories



Sim. threshold (DBSCAN only)	DBSCAN clusters		COBWEB clusters		
0.5	(a,b,c,d,e,f,g,h,i,j)		(a,b,c,d,e,f,g,h,i,j)		
0.7	(a,b,e,h,i,j)	(c,d,f)	(a,j,e,b,i,h)	(f,g,c,d,)	
0.9	(a,b)	(h,i,j)	(c,d,f)	(a,b,e)	(j,i,h)	(f,d,c)	(g)

Figure 5.2: Clustering Hierarchies produced by Thesus

are grouped together. This replacement reduces the number of concepts in the label to the desired one.

The output of the clustering module is the set of enriched documents, along with the cluster id to which each document belongs. The module also produces a label description, i.e., a set on concepts, for each cluster id.

Querying the results The clusters thus produced can be exploited to answer user queries, in a more meaningful way. They can also be used to locate documents in the collection by browsing. In Thesus, our motivation was to keep the user interface as simple and intuitive as possible, therefore we consider queries to be simple keyword based queries. Let $q = \{k_i\}$ represent a query, where k_i are keywords defining the users interest. Thesus will first of all identify the cluster(s) relevant to q. The irrelevant clusters are pruned and will not be considered in the search. The query-processing module then proceeds to determine within the selected cluster(s) the documents most relevant to the query, and can present the results ordered, and classified.

Another important feature of the query module, which is still undergoing development, is the ability to performed advanced queries for thematic hubs, authorities, co-citations and couplings. This is for the moment feasible only at the keyword level. However, we are working on employing link semantics in the search for strongly interconnected pages with similar semantics, and we want to try to integrate link structure into the clustering algorithm, and not just when constructing the set.

5.6 Experimental Results

In this section, we consider the performance of the whole Thesus system. Then, we evaluate the efficiency of the "information extraction" and clustering modules of Thesus. For the first module we choose URLs or sets of URLs whereas for the second we cluster a predefined (pre-classified) collection of URLs using the clustering algorithms and finally, evaluate the results. In the first set of experiments, we investigate the ability of incoming links semantics to characterize the contents of a page or a set of pages and illustrate the three different implementations of weighted group semantics extraction process. The second experiment tests the capabilities of the system in characterizing and clustering a set of URLs by comparing to the results of human clustering. Finally we present some results of the link analysis operators on a large experiment performed with Thesus. Let us note that in all our tests, we used blind testers (thus unbiased) to evaluate Thesus, and the other systems.

5.6.1 System performance

The first three modules of the Thesus architecture (information acquisition, extraction and enhancement) take as input an ontology and optionally an initial set of documents and produce a set of documents and links with keywords, semantics and the respective weights. This procedure happens once and creates the information pool for the Thesus clustering and querying modules.

The duration of the crawling procedure is mainly affected by the network bandwidth and the architecture of the crawling system itself as well as by the number of documents collected. In a prototype implementation, we used an ontology on music (containing approximately 100 terms), we started from 3 directory pages on music (those of Yahoo, DMOZ and Google directory) and crawled for 15 levels. The crawling was performed using a single computer running several threads to take advantage of the available bandwidth. The information acquisition took place simultaneously to the crawling. In a two days period, we collected 65000 documents.

The performance of the information enhancement module is affected by the size of each keyword set (one for each document) and by the size of the ontology. As the number of keywords in the set increases the number of combinations of senses that should be considered for each document increases. An increase on the number of terms in the ontology, increases the number of pairs (keyword, term) that should be computed. In the previous experiment the processing time for the whole collection was about 6 hours.

Technical note: Clustering is performed off-line, but is repeated several times with different parameters, until the best clustering schema is achieved. Clustering quality is validated using several internal measures. The time for clustering the set of documents into semantically coherent groups depends on the number of documents and on the algorithm used. Clustering can be accelerated by caching certain information that is calculated many times. In the current implementation, we pre-compute the similarity-matrix that contains the similarity between all pairs of documents in the set. For the music example, the size of this matrix in memory is (65000x65000)/2 bytes given that the similarity of two documents ranges from 0 to 100 and is stored as a byte number. This approximates to 1.97 Gigabytes of memory needed to store the matrix. Due to memory constraints, we clustered only a part of the document set (the 20000 documents with the most incoming links). The similarity matrix was computed in 365 seconds and the set was clustered, into 879 clusters in 86 seconds, using the DBSCAN algorithm.

The most crucial part of the system is the query module that must be

capable to answer users' queries fast. Query-response time depends on the type of the query. Typically, queries on thematic hubs and authorities are answered in less than 5 seconds, whereas normal matching queries are answered within 2 seconds. This is acceptable and we believe can still be improved with reasonable work.

5.6.2 Keyword extraction from incoming links - evaluation

Page characterization In order to demonstrate the capabilities of the information extraction module in characterizing the contents of a Web document, we selected 50 URLs and obtained their incoming links characterization using a maximum 100 incoming links for each URL. We aggregated results by source URL (`weightedSourceKeywords`) and kept the top 10 keywords for each URL. We presented this description, along with the descriptions provided by Altavista and Google to a group of blind testers and asked them to rate from 1 to 5 the quality of the description (1 - very bad, 5 - very good). In more than 50% of the cases Thesus descriptions were considered the best of the three, where as the average rating for Thesus results was 3.7 out of 5, outscoring the other two systems, Altavista - 1.9 and Google - 3.4. It is important to say that in some of the URLs used in the test, Google and Altavista descriptions either were manually generated (this is the case for pages in their directory) or contained the title of the page, whereas Thesus descriptions were automatically created and did not contain the title of the page.

Group characterization: semantics In order to test the usefulness of the operators that assign semantics to group of pages, based either on their incoming or outgoing links, we performed an experiment on the home pages of the VERSO research group at INRIA. The home pages where all listed in a single page (http://www-rocq.inria.fr/verso/members/members.html), which is a thematic coupling (See definition 5.4.13) to all of them.

If we call the set of the home pages URLs {U}, the set of pages that point to them {I} and the set of pages pointed by them {O} then we get the following results (limited to top):

- $weightedGroupKeywords(\{I\}, \{U\})$ = luc 81, home 57, authors 48, inria 47, serge 47, abiteboul 46, page 44, ioana 42, manolescu 42, voisard 33, back 31, grumbach 30, michel 23, cluet 22, university 22, sophie 22,

- $weightedTargetKeywords(\{I\}, \{U\})$ =home 15, page 13, inria 8, information 7, databases 6, university 6, Web 6, data 6, xml 6, authors 6, rocq 5, france 5, back 5,
- $weightedGroupKeywords(\{U\}, \{O\})$ = Web 26, inria 25, xml 22, page 22, verso 21, france 19, data 19, information 19, publications 17, paris 17, computer 16, science 16, research 16, database 15, hillebrand 14,
- $weightedSourceKeywords(\{U\}, \{O\})$ = inria 5, data 4, serge 3, semistructured 3, xml 3, abiteboul 3, time 2, logic 2, rick 2, vianu 2, comity 2, projets 2, databases 2, hull 2,

The first operator counts the number of times a keyword appears in the links of the first 100 pages that point to {U}. The second operator counts the number of pages that point to $\{U\}$ using a certain keyword. In order to find the incoming links of a page, we used Googles backward links service. The third operator counts the number of times a keyword appears in all the outgoings links of pages in $\{U\}$. The last operator counts the number of pages in $\{U\}$ that point to another page using a certain keyword.

Comparing the first two sets of keywords, we can easily see that keywords with a high value of TIMES in the first set do not appear in the second set. This is the case for keywords that appear many times but only in links to the same page (for example the names of the members appear often in the links to their home pages). Although, weightedGroupKeywords gives high values to these keywords, weightedTargetKeywords counts them only once, since they characterize only one page. The contents of the second set, characterize the group of URLs $\{U\}$ as a whole. As we already mentioned, all pages in $\{U\}$ have a common coupling that links to them. This leads to the fact that we can quickly characterize a set of pages that share a common coupling by characterizing the coupling itself.

The two latter sets of results inform on the interests of the VERSO members as those are expressed in the links they have in their home pages. In the last set we can see the words *inria*, which is the name of the research center, *serge* and *abiteboul* which is the name of the head of the group, *data*, *semistructured* and *xml* which are some of the main interests of the group. At this stage, perform more thorough semantic operations, we need to use, for instance, an ontology on *databases*.

Group characterization: Evaluation In this experiment, we extract using Thesus the keywords that characterize sets of pages that are listed under the same topic in Googles directory and measure their relevance to the topic, according to Thesus. The two first operators illustrated in the

previous example (weightedGroupKeywords and weightedTargetKeywords) are employed for the characterization, the N top keywords are extracted (N ranges from 3 to 10) and the number of keywords that are relevant to the topic is counted. The experiment is performed on 30 Google topics. For different values of N, we calculate the percentage of keywords that are relevant. The results are depicted in Figure 5.3.

The examples above, justify the notion that we already had, that Thesus operators allow the characterization of a set of pages based on incoming links in a satisfying level compared to characterization assigned by humans (which is about 80%). Another important fact is that aggregation of the results per target page (weightedTargetKeywords operator) yields better results than aggregation per keyword (weightedGroupKeywords). The above results indicate that as the number of keywords used for the characterization increases, the total relevance decreases and suggests keeping at most the 10 top keywords to characterize the set.

It is noteworthy that the results presented above used a maximum of 100 incoming links for each page in a set. Visiting 100 documents in order to characterize 1 is not very efficient, even when the documents have been pre-fetched or pre-processed. In another test, we attempt to characterize the same sets of pages as in the previous test, this time using fewer incoming links. In Figure 5.4 we illustrate how the number of incoming links affects the relevance of the extracted characterization to the topic. Only the top 7 keywords are taken into account. The figure shows that by reducing the amount of incoming links to 20, we still get reasonable results, less than 10% loss in accuracy in both operators.

5.6.3 Discovery thematic subsets

In order to test Thesus ability in characterizing and clustering a set of pages, we take a manually clustered set of URLs as the core set in Thesus and cluster it using our system. Then we measure the quality of the clustering scheme using an external quality measure, the entropy [132] as defined in [136].

The core set of pages contains the URLs suggested by DMOZ [117] under a few categories in the Music/Styles path. The categories are: hip-hop, experimental, world, blues, Indian music, pop, dance, filk, electronic, electronica, country, dance, polka, early music, new age, lounge, rhythm and blues, folk, classical, gamelan, opera, bluegrass, rock, easy listening and contain from 10 to 30 URLs. The total number of URLs is 481.

We measure the effectiveness of the two clustering algorithms implemented in Thesus, the DBSCAN and the COBWEB algorithms. The mini-

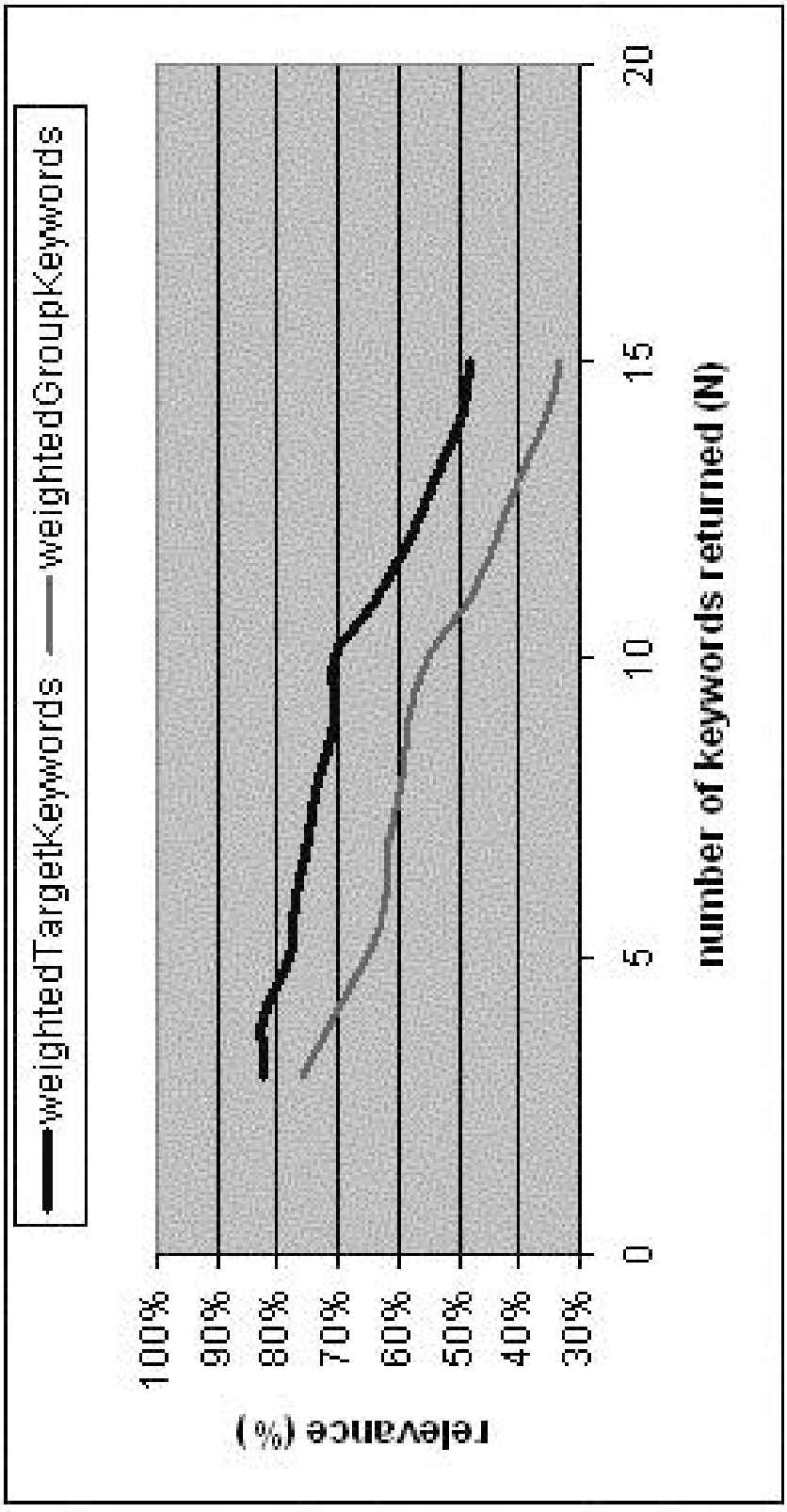


Figure 5.3: Relevance percentage group characterization

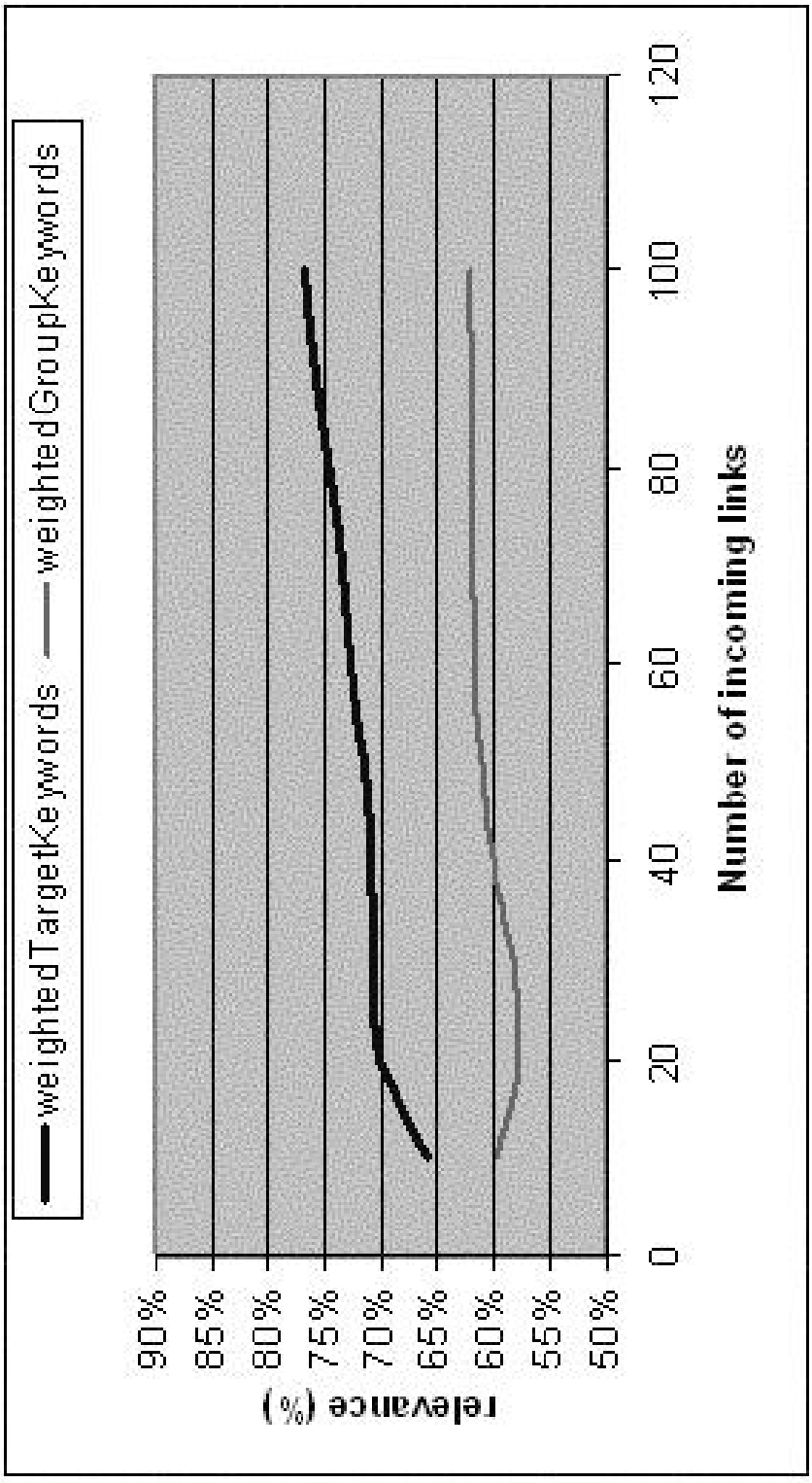


Figure 5.4: How relevance is affected by the number of incoming links

mum value of entropy is 0 and can be achieved by putting each document in a separate cluster. On the other hand, entropy reaches its maximum value when each cluster contains one document from each different category. In this case, the maximum value of entropy is log(N), where N is the number of different categories. In this experiment, we calculate the maximal entropy and compare it to the entropy of each clustering scheme using both DBSCAN and COBWEB for different values of N. The details of the produced clustering schemes are given in Tables **??**.

Docs	Categories	MinSim	Docs Clusered	Clusters	Entropy	Max entropy	Entropy decrease
400	23	0.55	115	8	0.96	1.36	29%
400	11	0.8	24	3	0.65	1.04	38%
191	22	0.55	74	8	0.76	1.34	43%
191	7	0.8	16	2	0.47	0.85	44%
457	27	0.65	132	6	1.13	1.43	21%
457	5	0.8	10	2	0.47	0.70	33%
182	23	0.6	71	7	0.79	1.36	42%
						Average decrease	36%

Table 5.1: Comparison of the DBSCAN results for various input parameters (minDocs=3) using concepts

Docs	Categories	MinSim	Docs Clustered	Clusters	Entropy	Maximum entropy	Entropy decrease
396	8	0.95	18	7	0.16	0.90	82%
396	13	0.85	31	4	0.67	1.11	40%
295	12	0.95	17	4	0.60	1.08	45%
295	20	0.8	48	6	0.90	1.30	31%
						Average decrease	49%

Table 5.2: Comparison of the DBSCAN results for various input parameters (minDocs=1) using concepts

Evaluation of the results We characterized documents by extracting keywords from the incoming links and enhancing information with seman-

Documents	Categories	Clusters	Entropy	Maximum entropy	Entropy decrease
25	2	7	0.11	0.30	63%
37	3	6	0.27	0.48	43%
55	4	12	0.22	0.60	64%
78	5	7	0.45	0.70	35%
100	10	33	0.58	1.00	42%
160	24	56	0.57	1.38	59%
244	26	85	0.59	1.41	58%
				Average decrease	52%

Table 5.3: COBWEB results (using concepts) for an increasing number of documents

tics using a manually created ontology on music (with 113 concepts) and WordNet.

In the DBSCAN case, documents clustered based first on semantics and then on the extracted keywords directly. If keywords have already been mapped to concepts, clustering based on semantics is significantly faster than clustering based on keywords, since the similarity between ontology concepts is calculated once for all the 113x113 combinations (using Wu and Palmers measure on the ontology), whereas the similarity between keywords must be calculated for every different combination of keywords that may appear. The latter calculation also involves accessing WordNet, which significantly slows down the process. In both cases, a significant decrease in the entropy of the system can be noticed, which is higher when documents are clustered based on the keyword characterization (49% of the maximum entropy, Table **??**) rather than using the respective semantics (36%, Table **??**).

In the COBWEB case, documents are clustered based on their semantics and the entropy of the clustering scheme is always lower than the maximum entropy (a decrease of 52%). As can be deduced from the results in Table **??**, the entropy seems to get stabilized for more than a certain number of documents (approximately 100). This means that the clustering algorithm needs several documents to be clustered in order to perform well.

5.6.4 Similarity measures

We can compare out similarity measure to the cosine similarity measure. On the one hand, documents are clustered using the Thesus similarity measure (THESIM) to compute document similarity. On the other hand, we cluster documents again using the keyword descriptions but this time using

the traditional cosine similarity measure (COSINE)[2]. First let us detail the experimental setup:

The test set: In the experiments we take the URLs that fall under the arts/music branch of DMOZ excluding the bands and artists sub-category, and all by-letter sub-categories, that mainly contain surnames, company names etc., with no conceptual content. This results to approximately 30.000 URLs from 2155 different categories. This is the first document categorization we compare with and that we name (FULL-SET).

We should stress here that the ontology used in this experiment is not identical to the part of the DMOZ tree from which we retrieved the test document set. This is done because terms in DMOZ paths do not necessarily appear in WordNet thus we cannot map extracted keywords to them and moreover we judge that the DMOZ topic hierarchy does not completely express a hierarchy of concepts related to music. For example, DMOZ may distinguish between Gamelan and Indian music, whereas in our ontology only the term World Music exists. In this case, it is expected that documents from both categories of DMOZ should be grouped together in our approach. Thus it is not expected that the clusters in our approach be exactly identical to the groups of documents classified to the DMOZ paths.

We create two more document sets with flattened categories. The first set (LEVEL1) contains the most populated sub-categories of arts/music. The categories contain approximately 25.000 documents and are listed in Table 5.4.

Sound files	1933
Instruments	6082
Lyrics	853
Vocal	733
Marching	928
Styles	13061
Electronic	378

Table 5.4: Number of documents per category in the LEVEL1 document set

The second set (LEVEL2) contains approximately 6000 documents under the path arts/music/instruments. Categories are flattened again. The number of documents for each of the categories is listed in the Table 5.5).

It is straightforward to see that $LEVEL2 \subset LEVEL1 \subset FULL\text{-}SET$.

[2]The cosine measure is simply the ratio $\frac{number of concepts two documents have in common}{number of distinct concepts in both documents}$

Electronic	378
Keyboard	784
Percussion	293
Squeezebox	94
Strings	2627
Winds	1735

Table 5.5: Number of documents per category in the LEVEL2 document set

Evaluating clustering quality using two external quality measures All the experiments are performed by applying different values to the input parameters of the DBSCAN algorithm. We evaluate clustering quality each time, using Rand Statistic and F-measure [130]. The highest values for both measures are depicted in the tables. The number of clusters produced in an experiment may differ significantly depending on the input parameters of the clustering algorithm. For each document set, description and similarity measure combination, we repeat the clustering process several times with different input parameters and compare the produced clustering scheme with the original categorization (2155 categories for the FULL-SET, 6 categories for LEVEL 1 and LEVEL 2). We keep the clustering scheme with the maximum likeness to the original categorization in respect to the F-measure and Rand Statistics values.

The two measures compute the covering between the produced clustering of a document set and a predefined categorization of the same set, which is considered as the baseline of comparison. In our case the baseline is the categorization of the documents by DMOZs editors. In general, external quality measures for clustering, such as F-measure and Rand Statistics, examine all the possible pairs of documents in a document set. The quality increases when documents of a pair belonging to the same category are clustered together and decreases when documents from the same category are assigned to different clusters.

- Input: Input parameters, MinSim is the minimum similarity between two documents in a cluster, MinDoc+1 is the minimum number of documents in a cluster.

- Output: The number of clusters produced and the number of documents clustered. The remaining documents are considered as noise.

- F-measure, Rand Stat.: The two clustering quality measures.

- Init. time: The time (in seconds) needed to calculate the distance between all pairs of documents.

- Cl. time: Average time (in seconds) for clustering the set of documents

As can been seen in the table, THESIM outperforms cosine similarity on keywords. The results achieved using incoming link descriptions are better than using most frequent keywords in the documents content and comparable to descriptions provided by humans. It is also evident that the computation of THESIM is not computationally expensive. Its execution time was slightly better that the COSINE measure computation.

A closer look on the figures of table 5.5 shows that the number of total documents for the 3 different description techniques differs significantly. This is because for each technique, the source of information is not available for all the documents in the set. For example, DMOZ provides descriptions for 22575 documents of the FULL-SET, whereas when we tried to access the documents and parse their contents, we only got information for only 12774 of them. The main reasons for this, is that either the documents were not accessible at the moment of the experiment or their contents could not be automatically parsed. On the other side, hyperlink information was more available, so we got information from incoming hyperlinks for 17913 of the documents in the set. The same variation occurs for the two other document sets (LEVEL 1, LEVEL 2).

Additionally when the descriptions that contain extracted keywords are replaced with descriptions containing concepts from the ontology, and the THESIM is used instead of the cosine similarity, the number of documents to be clustered decreases. The reason for this is that not all keyword descriptions are relevant to the domain of interest and as a consequence they are not mapped to ontology concepts. This illustrates the ability of the mapping mechanism to discard irrelevant documents before clustering.

5.6.5 Keyword versus concepts

The quality of results presented in Table 5.5 is influenced by two factors: the similarity measure used in each case and the use of concepts versus keywords. When using concepts instead of keywords, the similarity given by Thesus is higher than the cosine similarity. Keywords that are different be mapped to the same concept, thus increasing the similarity between two documents. In order to get an indication on the influence of the mapping process in the quality of clustering, we performed an additional experiment in which we used the dataset of LEVEL 2, with keyword descriptions extracted from the

pages content and the respective concepts. We used THESIM for sets of keywords, using WordNet instead of the domain ontology as a hierarchy. In this case, the computation of similarity between documents is much slower since WordNet must be accessed for every keyword and pre-processing is not feasible. The clustering process is also much longer to complete, since access time to WordNet is quite important. However as a consequence, clustering quality is slightly better when keywords are used. This is not surprising, since WordNet represents in a sense the universe of possibilities, and out ontology is merely an abstraction of this larger universe. The results are presented in Table 5.6.

		FULLSET		LEVEL1		LEVEL2	
		THESIM	COSINE	THESIM	COSINE	THESIM	COSINE
DMOZ	Input						
	MinDoc	1	1	5	5	5	5
	MinSim	1	0.4	0.7	0.2	0.65	0.2
	Output						
	Clusters	1120	2002	7	17	6	19
	Clust.doc.	16436	17921	14315	22967	2853	5577
	Tot. doc.	22575	29115	18252	23590	4615	5911
	F-measure	0.182	0.196	0.724	0.533	0.694	0.589
	Rand Stat	0.913	0.849	0.502	0.482	0.513	0.378
	Cl. time	80	90	40	60	0.5	0.5
	Init. Time	689	850	499	2285	25	78
CONTENT	Input						
	MinDoc	1	1	1	1	10	25
	MinSim	1	0.45	0.7	0.15	0.7	0.45
	Output						
	Clusters	598	1258	6	8	8	6
	Clust.doc.	9891	13507	10006	17545	1679	1663
	Tot. doc.	12774	21904	10242	17743	3011	4636
	F-measure	0.162	0.160	0.720	0.492	0.623	0.483
	Rand Stat	0.917	0.849	0.513	0.501	0.518	0.485
	Cl. time	52	60	40	60	0.60	0.6
	Init. Time	309	650	131	562	12	38
INLINKS	Input						
	MinDoc	1	1	5	5	5	5
	MinSim	1	0.4	0.65	0.1	0.65	0.1
	Output						
	Clusters	879	928	4	23	5	5
	Clust.doc.	12080	14397	11312	13534	2737	4062
	Tot. doc.	17913	18944	14248	15048	3473	4147
	F-measure	0.200	0.130	0.730	0.592	0.639	0.511
	Rand Stat	0.751	0.926	0.500	0.441	0.503	0.330
	Cl. time	88	93	52	60	2.7	3.1
	Init. time	365	565	580	598	34	385

Figure 5.5: Clustering results for the three datasets with optimal input parameters

	KEYWORDS	CONCEPTS
Input		
MinDoc	1	1
MinSim	1	0.45
Ouput		
Clusters	598	1258
Clustered Docs	9891	13507
Total Docs	12774	21904
F-Measure	0.162	0.160
Rand Stat	0.917	0.849
Clustering Time	52	60
Initialization Time	147	650

Table 5.6: Clustering in Thesus, using keyword and concept descriptions

5.6.6 Complexity analysis

Apart from the quality of the clustering, the efficiency and scalability of the system is important. The time required for clustering a set of Web documents is influenced by many factors such as, the number of documents, the number of concepts that describe each document, and the complexity of the clustering algorithm. The total time needed for clustering the set is also affected by the system parameters such as the CPU speed and the available amount of main memory.

Let N be the number of documents in the set and M the mean number of concepts for each document in the set. We also assume that the similarities between all pairs of concepts in the ontology have been pre-calculated and stored in main memory. The similarity between two documents using our measure is computable in $O(M^2)$assuming that the time to access the pre-calculated Wu and Palmer similarity for two concepts in the ontology is $O(1)$.

In [61], DBSCAN is used in the context of spatial databases and assumes that the elements to be clustered are points in a metric space, of a given dimension, usually 2 or 3. The distance measure used is a simple Euclidean distance. In order to compute the neighborhoods of a document, the points are inserted into an R*-Tree [24]. In our case, we are not in a metric space with known dimension. We cannot use an R*-Tree. Instead, we simply pre-calculate the similarity between all the N documents in the set. We save them in N different lists of size N. This pre-computation is in time $O(N^2)$. Then we sort each list, using Quicksort. Searching can be performed in $O(N \log N)$ on average, for a list of length N. To sort N lists, it takes time

$O(N^2 \log N)$. Once this pre-processing phase is complete, we can apply the algorithm.

The time complexity of the clustering procedure is based on the average complexity of defining density connected sets of documents, i.e., identifying the neighbors of the documents in the database. In our system, since the similarity of a document d_i with all other documents is pre-calculated in an ordered list, the time complexity of defining the list of the documents closest to it (i.e., with $\zeta > MinSim$) using a divide and conquer method is $O(\log N)$. The algorithm does this task once for every document in the set, thus the complexity is $O(N \log N)$.

5.6.7 Link analysis operators

A larger scale experiment has also been performed that collected and characterized approximately 40.000 Web pages related to technology. There were approximately 700.000 documents that pointed to at least one document in the core set, and that we parsed in order to construct link semantics for the set of 40.000. Therefore, all these documents were fetched and the information carried by their links was extracted and stored in the database. Over 1.7 million links were stored.

We present here a few significant results that illustrate the use of link analysis operators (thematic hubs, authorities etc).

Thematic Authorities In the following, we present the top three results of the query "nuclear physics research" with a minimum number of 5 incoming links, ranked (in our case) according to the number of incoming links. The results provided by the other search engines ([70, 139, 164]) for the same query, are ranked according to their methods. Results provided by Thesus are quite comparable those of search engines. This moreover shows that popular Web search engines favor authorities and hubs.

Thematic Hubs We ran the same query as previously, but this time searching for hubs. We considered as hub a page with more than 5 outgoing links with the semantics "nuclear physics research". We rank our results by the number of outgoing links. This functionality does not exist with current search engines, since in a way it is redundant with their role. Thesus output the following results:

- (18 links) www.cern.ch/Physics/HEP.html Over 200 links to various Nuclear Physics Centers. A great hub.

Thesus	18 links	www.cern.ch/
		European Organization for Nuclear Research
	10 links	www.rarf.riken.go.jp/rarf/np/
		A hub on nuclear physics pages
	7 links	www.er.doe.gov/production/henp/henp.html
		The US office of High Energy and Nuclear Physics.
Google	1	www.cern.ch/
	2	www.rarf.riken.go.jp/rarf/np/nplab.html
	3	www.rcnp.osaka-u.ac.jp/index-e.html
		Japanese Research center for nuclear physics
Teoma	1	www.iop.org/
		The Institute of Physics
	2	www.pppl.gov/
		Princeton Plasma Physics Laboratory (PPPL)
	3	www.rarf.riken.go.jp/rarf/np/nplab.html
Yahoo	1	www.er.doe.gov/production/henp/henp.html
		The US Office of High Energy and Nuclear physics.
	2	www.jinr.dubna.su/
		Russian Joint Institute for Nuclear Research
	3	www.nikhef.nl/
		Dutch National Institute for Nuclear Physics and High Energy Physics

Table 5.7: Thematic Authorities of Thesus

- (9 links) www.search-beat.com/Science/Technology/Energy/Nuclear/ About 75 links concerning Nuclear Physics Research. Good hub.

- (6links) netmation.com/www/phystd.htm: The page concerning the Physics links. Over 100 Physics links. Fewer Nuclear Physics links

Thematic co-citations We then search for pages that are pointed by every one of the three hubs on nuclear physics research, using terms research and institute in the hyperlink semantics. The 18 results (Appendix **??**) contained: a) 10 home pages of institutes that perform research in nuclear or high energy physics, b) 4 home pages of universities or institute of general scope, c) 2 home pages of institutes not strongly related to nuclear physics, d) 2 hub pages in nuclear physics.

This example shows that Thesus has a good capacity of finding thematic hubs on the Web and on exploiting this to locate highly co-cited pages that share common semantics. This is a feature that is not treated adequately by regular search engines. Also, when browsing the pages returned as hubs, we noticed that there were always more links about a given topic than we had discovered. This strengthens our idea that pages that are pointed by the same page often share common semantics.

5.7 Conclusion

When comparing our results on various datasets ranging from a few hundreds to a few thousands of pages to results provided by Web search engines or Web directories, we come to the following conclusions. Querying links semantics yields correct information on the semantics of pages, at least comparable to those established with full text querying. We note that is is also possible to accurately rank pages by the number of incoming links of a given semantics that they have, rather than simply the number of page pointing to them. Furthermore, the discovery of thematic hubs is an easy, powerful tool for finding information on the Web, and their use in complex queries yield impressive results.

And finally, defining a measure of similarity between Web pages, which is based on semantics conveyed by links and on the connectivity between pages, lets us discover clusters of pages that have similar semantics and have similar connectivity features.

Web search engines answer users queries with a set of pages that match the majority of terms in the query. Information retrieval algorithms rank the results based on the distance of the query to the document contents.

Yet, document contents have been created by the document authors, who in some cases are interested by getting a high rank for their pages. Such algorithms can be manipulated by page authors by adding text which can affect search engines ranking for a query. With hyperlink information, the introduction of such bias becomes harder. Searching in the World Wide Web is a very important task, in social and financial terms, as hundreds of millions of users worldwide, with diverse profiles, are searching for information on the Web. We believe that they can be helped in that task much beyond what in provided by current Web search engines.

We have shown how to capitalize on this observation and we present Thesus model and language, which can be applied for selecting, processing and querying the World Wide Web. The distinguishing features of Thesus language are a. it extracts and exploits additional semantics from links b. it enables queries and organization of pages based on aggregate connectivity features and link semantics. We also present the implemented Thesus system that collects URLs based on a given set of keywords, extracts information of the collections incoming and outgoing links (by processing links neighboring text in the source URL). The system enhances extracted information with semantics, using an ontology and a thesaurus and populates a relational database with all this information. A clustering module is able to detect subsets of the initial document set that have similar semantics, assigned by incoming links. An additional client application allows complex queries based on connectivity semantics.

Chapter 6

Conclusion

In this thesis, we have studied the definition, architecture and specific features of a Web Warehouse. We have shown that it is possible to design in a simple way a Web Warehousing application, using the declarative Spin specification language, and by using the functionalities of Web services, both simple services such as the crawling and fetching of pages, and more complex ones such as semantic enhancement and clustering. In our work, we bore in mind two key aspects of applications focused on Web data : genericity and scalability.

In particular, two algorithms we developed for the specific needs of Xyleme and Thesus can be used in a general, large scale context. The Xyleme monitoring algorithm can be used by high speed processing applications that need to process patterns that are conjunctions of atomic elements. The Thesus clustering module will function on any collection of documents characterized by a short set of keywords, and an ontology. Both algorithms have been tested in real world Web application scenarii, and they have proven functional.

We can make a general comment about this work. The information available on the Web is massive, yet we have little *a priori* knowledge of it. As a consequence, the objectives have to remain modest. This is unfortunately inherent to our object, the World Wide Web, at least in its current state.

Regarding experimentation, we did our best to prove sturdy results, but in many cases, experimentation is subjective. For instance, it is not simple to obectively assess the relevance of a page to a query. We do nonetheless believe that the use of blind testers, i.e., humans that subjectively mark the results, without knowing by which system they have been produced generate unbiased testing.

Perspectives : The Data retrieved on the Web bears serious limitations : it is loosely structured, and conducting a precise analysis, in the way relational data could be processed, is impossible. This is one of the reasons we studied methods of semantic extraction and categorization. In the case where the pages returned have structural similarities (e.g., XML pages with the same XMLSchema), it should be possible to apply much deeper processing.

The Spin system is a first step in this direction. By using a semi-structured model of data, it proposes a simple way of integrating data from the Web into a single warehouse, on which specific complex applications may be run. This is one of the goals of the e.dot project.

Finally, we believe that with the emergeance of the Semantic Web, where more importance is attached to the description and relevance of documents, and an understanding of their semantics, one can go further in discovering, analyzing and integrating Web information.

Chapter 7

Personal Bibliography

Journal Papers

- M. Halkidi, B. Nguyen, I. Varlamis and M. Vazirgiannis, *Thesus: Organising Web Document Collections based on Semantics and Clustering*, *Very Large Databases Journal*, special edition on Semantic Web, 2003.
- I. Varlamis, M. Vazirgiannis, M. Halkidi and B. Nguyen, *Thesus, A Closer View on Web Content Management Enhanced with Link Semantics*, to appear in Transaction on Knowledge and Data Engineering
- Lucie Xyleme, *A dynamic warehouse for XML data of the Web* in IEEE Data Engineering Bulletin, 2001

International Conferences

- B. Nguyen S. Abiteboul, G. Cobena and M. Preda, *Monitoring XML data on the Web*, Proceedings of the ACM-SIGMOD, 2001

International Workshops

- B. Nguyen, S. Abiteboul, G. Cobena and L. Mignet, *Query Subscription in an XML Warehouse*, DELOS Workshop, 2000

European Demos

- S. Abiteboul, V. Bensal, G. Cobena, B. Nguyen and A. Poggi *Model, design and construction of a Service-Oriented Web Warehouse*, in European Conference on Digital Libraries, 2003

Poster Presentations

- B. Nguyen, M. Vazirgiannis, I. Varlamis and M. Halkidi *Thesus: Effective Thematic Selection, Labeling and Organization of Web Document Collections Based on Link Semantics*, in European Conference on Digital Libraries, 2003

French Conferences and Demos

- S. Abiteboul, G. Cobena, B. Nguyen, A. Poggi, *Construction Sets of Pages of Interest*, Bases de Données Avancées, 2002
- B. Nguyen, M. Vazirgiannis, I. Varlamis, M. Halkidi, *Organising Web Documents into Thematic Subsets using an Ontology (Thesus)* , in Journées AS-Web Semantique, 2002
- B. Nguyen, M. Vazirgiannis, I. Varlamis, M. Halkidi, *Construction de classes de documents Web*, in Journées Francophones de la Toile, 2003
- B. Nguyen, M. Vazirgiannis, I. Varlamis, M. Halkidi, *A System for Creating and Organizing Web Documents Collections Using an Ontology (THESUS)*, in Bases de Données Avancées, 2003 (demo)

Appendix A

Behavior of $START_{N,h}(i)$

To calculate $START_{N,h}(i)$, we have to count the number of complex events P that start with a given atomic event a_i, and the total number of complex events Q. Clearly, $Q = \binom{N}{h}$ and P represents the number of ways to select $h-1$ elements in a set of $N-i$ elements, i.e., $P = \binom{N-i}{h-1}$. Hence: $START_{N,h}(i) = \frac{\binom{N-i}{h-1}}{\binom{N}{h}}$. This can be rewritten in the following formula:

$$START_{N,h}(i) = \frac{1}{(h-1)! \times \binom{N}{h}} \times \prod_{j=0}^{h-2}(N-j-i).$$

Therefore, the following relations hold:

$$\begin{array}{l} \forall h > 2, (i,N) \in \mathbb{N}^{*2}, i+h < N \\ \text{we have} \\ \prod_{j=0}^{h-2}(1-\frac{i}{N-j}) \quad < \quad (1-\frac{i}{N})^{h-1} \text{and} \\ \prod_{j=0}^{h-2}(N-j) \quad < \quad N^{h-1} \end{array}$$

We have:

$$\begin{aligned} START_{N,h}(i) &= \frac{\prod_{j=0}^{h-2}(N-j)}{(h-1)! \times \binom{N}{h}} \times \prod_{j=0}^{h-2}(1-\frac{i}{N-j}) \\ START_{N,h}(i) &= \frac{(\prod_{j=0}^{h-2}(N-j)) \times h!(N-h)!}{(h-1)!N!} \\ &\quad \times \prod_{j=0}^{h-2}(1-\frac{i}{N-j}) \\ START_{N,h}(i) &= \frac{h!(N-h)!}{(h-1)!(N-h+1)!} \times \prod_{j=0}^{h-2}(1-\frac{i}{N-j}) \end{aligned}$$

$$START_{N,h}(i) = \frac{h}{N-h+1} \times \prod_{j=0}^{h-2}(1-\frac{i}{N-j})$$

Since $h << N$, we can write the result simply as:

$$START_{N,h}(i) = \mathcal{O}(\frac{h}{N}(1-\frac{i}{N})^{h-1})$$

This proves Lemma 4.5.2.

Appendix B

Existance of table H_i

In this appendix, we prove show that it is reaseonable to follow Assumption 4.5.4.

The exact value for $LAYER^1_{N,h,M}(i)$ The exact calculation for $LAYER^1_{N,h,M}(i)$, if we do not accept the possibility of generating twice the same atomic event is:

$$\begin{aligned} LAYER^1_{N,h,M}(i) &= \frac{hash\ trees\ of\ size\ M\ w/o\ table\ H(i)}{total\ nr\ hash\ trees\ of\ size\ M} \\ LAYER^1_{N,h,M}(i) &= \frac{(i-1)\binom{2^{N-1}}{M} + \binom{2^{N-i}}{M}}{\binom{2^N}{h}} \end{aligned}$$

This value is complicated to compute, this is why we use the approximate value that follows. If we do not exclude the possibility that some of these complex events may be identical, it is easy to see that: $LAYER^1_{N,h,M}(i) = 1-(1-START_{N,h}(i))^M$. We will now prove that the probability of generating the same complex events is negligeable, and that the formula just given can be approximated by 1 in most cases. In any case, by saying that the table exists, we are just making a pessimistic assumption.

Probability of generating the same complex events Of course, in our 'live' application, we will have exactly M *distinct* complex events. We argue here that given the figures of 'live' applications, the probability of generating two identical complex events is very small. Let us calculate this probability. The probability of generating the same complex event after having generated x distinct ones is: $P_{identical}(N,h,x) = \frac{x}{\binom{N}{h}}$. We can easily

calculate the probability of generating M distinct complex events, it is:

$$\begin{aligned} P_{distinct}(N,h,M) &= \prod_{i=1}^{M-1}(1-i/\binom{N}{h}) \\ P_{distinct}(N,h,M) &\geq (1-\frac{M}{\binom{N}{h}})^M \end{aligned}$$

We give a numerical application: $N = 10^6, M = 10^7, h = 4$, with $\binom{N}{h} \approx N^h = 10^{24}$ thus $P_{distinct} \geq (1-10^{-17})^{10^7}$, $P_{distinct} \geq 1-10^{-10}$. This shows that neglecting the fact we can generate two identical complex events is a very pertinent assumption, given the range of values we have[1].

[1] This comes from the fact that with our values of N and h, $\binom{N}{h} = \mathcal{O}(\frac{N^h}{h!}) \approx N^h$, which in our case is $\Omega(M)$.

Appendix C

Demonstration of Lemma 4.5.3

The demonstration is as follows. $\forall N, h, i \in (\mathbb{N}^*)^3, (h+i+1) < N$ we have:

$$\begin{aligned} \frac{\binom{N-1}{h-1}}{N-1} &= \frac{(N-1)!}{(h-1)! \times (N-h)! \times (N-1)} \\ \frac{\binom{N-1}{h-1}}{N-1} &= \frac{(N-2)!}{(h-1)! \times (N-h)!} \end{aligned}$$

and

$$\begin{aligned} \frac{\binom{N-i}{h-1}}{N-i} &= \frac{(N-i)!}{(h-1)! \times (N-h+1-i)! \times (N-i)} \\ \frac{\binom{N-i}{h-1}}{N-i} &= \frac{(N-i-1)!}{(h-1)! \times (N-h+1-i)!} \end{aligned}$$

We see that:

$$\frac{\binom{N-1}{h-1}}{N-1} = \frac{\binom{N-i}{h-1}}{N-i} \times \lambda$$

where

$$\begin{aligned} \lambda &= \prod_{j=1}^{i} \frac{N-1-j}{N-1-h-j} \\ \lambda &\geq 1 \end{aligned}$$

This proves Lemma 4.5.3

Bibliography

[1] S. Abiteboul, V. Bansal, G. Cobena, B. Nguyen, and A. Poggi. Model, design and construction of a service-oriented web-warehouse (demo). In *Europeean Conference on Digital Libraries*, 2003. Trondheim, Norway.

[2] S. Abiteboul, O. Benjelloun, I. Manolescu, T. Milo, and R. Weber. Active XML: Peer-to-Peer Data and Web Services Integration (demo). In *Very Large Databases*, 2002.

[3] S. Abiteboul, O. Benjelloun, T. Milo, I. Manolescu, and R. Weber. Active xml: A data-centric perspective on web services. In *Bases de Données Avancées*, 2002.

[4] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web*. Morgan Kaufmann, California, 2000.

[5] S. Abiteboul, G. Cobena, B. Nguyen, and A. Poggi. Construction and maintenance of a set of pages of interest (spin). In *Bases de Données Avancées*, 2002.

[6] S. Abiteboul, M. Preda, and G. Cobena. Computing web page importance without storing the graph of the web. *IEEE Data Engineering Bulletin*, 1(25):27–33, 2002.

[7] S. Abiteboul, M. Preda, and G. Cobena. Computing web page importance without storing the graph of the web. *IEEE-CS DataEngineering Bulletin*, 25(1):27–33, 2002.

[8] S. Abiteboul, M. Preda, and G. Cobena. Adaptive on-line page importance computation. In *International World Wide Web Conference*, 2003.

[9] S. Abiteboul and V. Vianu. Queries and computation on the web. *ICDT*, 1997.

[10] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, pages 487–499. Morgan Kaufmann, 12–15 1994.

[11] V. Aguilera. X-OQL query language for XML. `http://www-rocq.inria.fr/ aguilera/xoql/index.html`.

[12] V. Aguilera, S. Cluet, P. Veltri, and F. Watez. Querying XML documents in Xyleme. In *ACM SIGIR Workshop on XML and information retrieval*, 2000.

[13] R. Al-Halami and R. Berwick and. Ch. Fellbaum editor. *WordNet, an electronic lexical database*. Bradford Books, may 1998. ISBN 0-262-06197-X.

[14] R. Albert, H. Jeong, and A.L. Barabasi. Internet: Diameter of the World Wide Web. *Nature*, 401:130–131, 1999.

[15] S. Alexaki, N. Athanasis, V. Christophides, G. Karvounarakis, A. Maganaraki, D. Plexousakis, and K. Toll. Rdfsuite: High level scalable tools for the semantic web. *ERCIM News*, (51), 2002.

[16] P.G. Anick and S. Vaithyanathan. Exploiting clustering and phrases for context-based information retrieval. In *Proceedings of the SIGIR Conference*, 1997.

[17] Apache web server. http://www.apache.org/.

[18] The internet archive. http://www.archive.org/.

[19] R. Armstrong, D. Freitag, T. Joachims, and T. Mitchell. Web watcher: A learning apprentice for the world wide web.

[20] G. Arocena, A. Mendelzon, and G. Mihaila. Applications of a web query language. In *International World Wide Web Conference*, number 6, 1997.

[21] G.O. Arocena, A.O. Mendelzon, and G.A. Mihaila. Applications of a Web query language. *Computer Networks and ISDN Systems*, 29(8–13):1305–1315, 1997.

[22] K. Atkinson. Mysql++ a c++ api for mysql, 2000. http://www.mysql.com/documentation/.

[23] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval.* ACM Press, 1999.

[24] N. Beckmann, H.P. Kriegel, R. Schneider, and B. Seeger. The r*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the ACM-SIGMOD*, 1990.

[25] K. Bharat and M.R. Henzinger. Improved algorithms for topic distillation in a hyperlinked environment. In *Proceedings of SIGIR-98, 21st ACM International Conference on Research and Development in Information Retrieval*, pages 104–111, Melbourne, AU, 1998.

[26] A. Bidault, B.Safar, and Ch. Froidevaux. Proximite entre requetes dans un contexte mediateur. In *Reconnaisance de Formes et Intelligence Artificielle*, 2002.

[27] A. Bonifati, D. Braga, A. Campi, and S. Ceri. Active xquery. In *Internation Conference on Data Engineering*, 2002.

[28] A. Bonifati and S. Ceri. Comparative analysis of five xml query languages. *SIGMOD Record*, (29), 2000.

[29] O. Brandman, J. Cho, H. Garcia-Molina, and N. Shivakumar. Crawler-friendly web servers. In *Proceedings of the Workshop on Performance and Architecture of Web Servers (PAWS)*, Santa Clara, California, 2000.

[30] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *International World Wide Web Conference*, number 7, 1998.

[31] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener. Graph structure in the web. In *9th International World Wide Web Conference*, 2000.

[32] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, and R. Stata. Graph structure in the web. In *Proceedings of the 9th International World Wide Web Conference,*, pages 247–256, 2000.

[33] P. Buneman, S. Davidson, W. Fan, C. Hara, and W.C. Tan. Keys for XML. *Computer Networks, Vol. 39*, August 2002.

[34] S. Chakrabarti, M. Berg, and B. Dom. Focused crawling: A new approach to topic-specific web resource discovery. In *International World Wide Web Conference*, number 8, may 1999.

[35] S. Chakrabarti, B. Dom, D. Gibson, J. Kleinberg, P. Raghavan, and S. Rajagopalan. Automatic resource list compilation by analyzing hyperlink structure and associated text. In *International World Wide Web Conference*, number 7, 1998.

[36] S. Chakrabarti, M. van den Berg, and B. Dom. Focused crawling: a new approach to topic-specific Web resource discovery. *Computer Networks (Amsterdam, Netherlands: 1999)*, 31(11–16):1623–1640, 1999.

[37] S. Chaudhuri and U. Dayal. Data warehousing and olap for decision support. *VLDB Tutorial*, 1996.

[38] S. Chaudhuri and U. Dayal. An overview of data warehousing and OLAP technology. *ACM SIGMOD Record*, 26(1):65–74, 1997.

[39] S. Chawathe, S. Abiteboul, and J. Widom. Representing and querying changes in semistructured data. *Proceedings of the IEEE International Conference on Data Engineering*, pages 4–13, 1998.

[40] S. Chawathe, S. Abiteboul, and J. Widom. Managing historical semistructured data. *Theory and practice of object systems*, 5(3):143–162, August 1999.

[41] J. Chen, D. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: A scalable continous query system for the internet databases. In *Proceedings of the ACM-SIGMOD*, page 379, 2000.

[42] J. Cho and H. Garcia-Molina. The evolution of the web and implications for an incremental crawler. In *Proceedings of the Twenty-sixth International Conference on Very Large Databases*, 2000.

[43] J. Cho and H. Garcia-Molina. Parallel crawlers. In *Proc. of the 11th International World–Wide Web Conference*, 2002.

[44] J. Cho, H. Garcia-Molina, and L. Page. Efficient crawling through URL ordering. *Computer Networks and ISDN Systems*, 30(1–7):161–172, 1998.

[45] G. Cobena. *Change Management of Semi-Structured Data on the Web.* PhD thesis, Ecole Polytechnique, 2003.

[46] G. Cobena. *Change management on semi-structured data from the Web.* PhD thesis, Ecole Polytechnique, 2003.

[47] G. Cobena, S. Abiteboul, and A. Marian. XyDiff, tools for detecting changes in XML documents. `http://www-rocq.inria.fr/~cobena/XyDiffWeb/`.

[48] G. Cobena, S. Abiteboul, and A. Marian. Detecting changes in xml documents. In *Internation Conference on Data Engineering*, 2002.

[49] WebCQ, OpenCQ webpage. http://www.cc.gatech.edu/projects/disl/WebCQ/.

[50] Corba web page. http://www.omg.org/.

[51] The darpa agent markup language ontology library. http://www.daml.org/ontologies/.

[52] U. Dayal, B. Blaustein, A. Buchmann, U. Chakravarthy, M. Hsu, R. Ledin, D. McCarthy, A. Rosenthal, S. Sarin, M.J. Carey, M. Livny, and R. Jauhari. The hipac project: Combining active database and timing constraints. *ACM SIGMOD Record*, 17(1):51–70, 1988.

[53] C. Delobel, C.Reynaud, M.C. Rousset, J.P. Sirot, and D. Vodislav. Semantic integration in xyleme: a uniform tree-based approach. *Data and Knowledge Engineering Review*, 2(44), 2001.

[54] E. Desmontils and C. Jacquin. *Indexing a Web Site with a Terminology Oriented Ontology in The Emergind Semantic Web*. I.F. Cruz and S. Decker and J. Euzenat and D.L. McGuinness, 2002.

[55] C. Ding, X. He, P. Husbands, H. Zha, and H. Simon. PageRank, HITS and a unified framework for link analysis. In *Proceedings of the ACM-SIGIR*, pages 353–354, 2002.

[56] A. Doan, J. Madhavan, P. Domingos, and A. Halevy. Learning to map between ontologies on the semantic web, 2002.

[57] M. Doherty, R. Hull, and M. Rupawalla. Structures for manipulating proposed updates in object-oriented databases. In *Proceedings of the ACM-SIGMOD*, pages 306–317, 1996.

[58] Document object model (DOM) level 1 specification version 1.0, October 1998.

[59] http://www-rocq.inria.fr/verso/edot/.

[60] T. Eiter and H. Mannila. Distance measures for point sets and their computation. *Acta Informatica Journal*, (34), 1997.

[61] M. Ester, H.P. Kriegel, J. Sander, and X. Xu. A density based algorithm for discovering clusters in large spatial databases with noise. In *International conference on Knowledge Discovery and Data Mining ACM-SIGKDD*, number 2, 1996.

[62] F. Fabret, F. Llirbat, J. Pereira, and D. Shasha. Publish/subscribe on the web at extreme speed. In *Very Large Databases*, 2000.

[63] A. Farquhar, R. Fikes, and J. Rice. The ontolingua server: A tool for collaborative ontology construction, 1996.

[64] D. Fensel. Ontologies: Silver bullet for knowledge management and electronic commerce.

[65] D. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine learning*, 2:139–172, 1987.

[66] G. Flake, S. Lawrence, and C.L. Giles. Efficient identification of web communities. In *Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 150–160, Boston, MA, August 20–23 2000.

[67] D. Florescu, A.Y. Levy, and A.O. Mendelzon. Database techniques for the world-wide web: A survey. *SIGMOD Record*, 27(3):59–74, 1998.

[68] D. Gibson, J.M. Kleinberg, and P. Raghavan. Inferring web communities from link topology. In *UK Conference on Hypertext*, pages 225–234, 1998.

[69] A. Gionis, D. Gunopulos, and N. Koudras. Efficient and tunable similar set retrieval. In *Proceedings of the ACM-SIGMOD*, 2001.

[70] Google internet search engine. http://www.google.com/.

[71] Google. Google Web APIs. `www.google.com/apis/`.

[72] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *J. Data Mining and Knowledge Discovery*, 1(1):29–53, 1997.

[73] J. Green, N. Horne, E. Orlowska, and P. Siemens. A rough set model of information retrieval. *Theoretica Infomaticae*, (28):273–296, 1996.

[74] N. Guarino. Formal ontology and information systems. In N. Guarino, editor, *International Conference on Formal Ontology in Information Systems*, number 1. IOS Press, june 1998. Trento, Italy.

[75] M. Halkidi, Y. Batistakis, and M. Vazirgiannis. On clustering validation techniques. *Intelligent Information Systems Journal*, 2001.

[76] M. Halkidi, B. Nguyen, I. Varlamis, and M. Vazirgianis. Thesus: Organising web document collections based on semantics and clustering. Technical Report, obtainable at ftp://ftp.inria.fr/INRIA/Projets/verso/VersoReport-214.ps.gz, 2002.

[77] M. Halkidi, B. Nguyen, I. Varlamis, and M. Vazirgiannis. Thesus: Organizing web document collections based on semantics clustering. *Very Large Databases Journal*, 2003. Special Edition on Semantic Web.

[78] E. Hanson, C. Carnes, L. Huang, M. Konyala, L. Noronha, S. Parthasarathy, J.B. Park, and A. Vernon. Scalable trigger processing. In *Proceedings of the 15th International Conference on Data Engineering*, pages 266–275, 1999.

[79] T. Haveliwala, A. Gionis, and P. Indyk. Scalable techniques for clustering the web. In *WebBD workshop*, 1998.

[80] X. He, H. Zha, C.H.Q. Ding, and H.D. Simon. Web document clustering using hyperlink structure. *Computational Statistics and Data Analysis*, 41:19–45, 2002.

[81] M. Henzinger. Hyperlink analysis for the web. *IEEE Internet Computing*, jan-feb 2001.

[82] Jalios. `http://www.jalios.com`.

[83] J.Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, (46), 1999.

[84] J. Jouglet. Souscription de requêtes dans un entrepôt de données xml. Stage d'option scientifique de l'École Polytechnique, 2000.

[85] The kartoo system. http://www.kartoo.fr/.

[86] G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis, and M. Scholl. RQL: A declarative query language for RDF. In The 11th Intl. World Wide Web Conference (WWW2002).

[87] M.M. Kessler. Bibliographic coupling between scientific papers. *American Documentation*, 14(1), 1963.

[88] M. Kobayashi and K. Takeda. Information retrieval on the web. *ACM Computing Surveys*, 32(2):144–173, 2000.

[89] C. Lagoze. The Warwick Framework : a container architecture for diverse sets of metadata. *D-Lib Magazine*, 1996.

[90] S. Lawrence and C.L. Giles. Searching the World Wide Web. *Science*, 280(5360):98–100, 1998.

[91] Y. Li. Towards a qualitative search engine. *IEEE Internet Computing*, pages 2–7, 1998.

[92] D. Lin. An information-theoretic definition of similarity. In *International Conference on Machine Learning*, number 15, 1998.

[93] L. Liu, C. Pu, and W. Tang. Continual queries for internet scale event-driven information delivery. *IEEE Transactions on Knowledge and Data Engineering*, 11(4):610, 1999.

[94] L. Liu, C. Pu, W. Tang, and W. Han. Conquer: A continual query system for update monitoring in the WWW. *International Journal of Computer Systems, Science and Engineering*, 2000.

[95] S. Luke, L. Spector, D. Rager, and J. Handler. Ontology-based web agents. In W. Lewis Johnson and Barbara Hayes-Roth, editors, *Proceedings of the First International Conference on Autonomous Agents (Agents'97)*, pages 59–68, Marina del Rey, CA, USA, 1997. ACM Press.

[96] G. Marchionini. Information seeking in electronic environments, 1995.

[97] M. Marchiori. The quest for correct information on the web: Hyper search engines. In *Proceedings of the Sixth International Conference on the World Wide Web*, 1997.

[98] A. Marian, S. Abiteboul, G. Cobena, and L. Mignet. Change-centric Management of Versions in an XML Warehouse. In *Very Large Databases*, 2001.

[99] A.O. Mendelzon, G.A. Mihaila, and T. Milo. Querying the world wide web. *Int. J. on Digital Libraries*, 1(1):54–67, 1997.

[100] Dublin Core Metadata.

[101] A.D. Mezaour. Focused search on the web using wequel, in proceedings of the 10th international workshop on knowledge representation meets databases (krdb 2003), hamburg, germany, september 15-16, 2003. In François Bry, Carsten Lutz, Ulrike Sattler, and Mareike Schoop, editors, *CEUR Workshop Proceedings*, volume 79. Technical University of Aachen (RWTH), 2003.

[102] A. Michard. *XML, langage et applications*. Eyrolles, Paris, 1999.

[103] L. Mignet, S. Abiteboul, S. Ailleret, B. Amann, A. Marian, and M. Preda. Acquiring XML pages for a webhouse, October 2000. in Bases de Données Avancées.

[104] L. Mignet, V. Aguilera, S. Ailleret, and P. Veltri. Xyro: The xyleme robot architecture. In *DIWeb*, pages 91–99, 2001.

[105] L. Mignet, D. Barbosa, and P. Veltri. The xml web: a first study. In *World Wide Web Conference*, 2003.

[106] Mind-it web page. http://mindit.netmind.com/.

[107] G. Moerkotte. The AODB relational system. U. Mannheim, personal communication, 1999.

[108] B. Nguyen, S. Abiteboul, G. Cobena, and L. Mignet. Query subscription in an xml webhouse. In *DELOS Workshop: Information Seeking, Searching and Querying in Digital Libraries*, 2000. Zurich, Switzerland.

[109] B. Nguyen, S. Abiteboul, G. Cobena, and M. Preda. Monitoring xml data on the web. In *Proceedings of the ACM-SIGMOD*, 2001.

[110] B. Nguyen, I. Varlamis, M. Halkidi, and M. Vazirgiannis. Construction de classes de documents web. In *Journées Francophones de la Toile*, pages 173–182, 2003.

[111] B. Nguyen, I. Vazirgiannis, I. Varlamis, and M. Halkidi. Organising web documents into thematic subsets using an ontology. In *Journées Action Spécifique Web Sémantique*, October 2002. Paris, France.

[112] B. Nguyen, M. Vazirgiannis, I. Varlamis, and M. Halkidi. A system for creating and organizing web document collections using an ontology, 2003.

[113] B. Nguyen, M. Vazirgiannis, I. Varlamis, and M. Halkidi. Thesus : Organising web document collections based on links semantics, poster. In *Europeean Conference on Digital Libraries*, August 2003. Trondheim, Norway.

[114] Niagara webpage. http://www.cs.wisc.edu/niagara/.

[115] I. Niiniluoto. *Truthlikeness*. D. Reidel Publishing Company, 1987. Dordrecht, Holland.

[116] Northern light news search. http://www.northernlight.com/news.html.

[117] Odp - open directory project. http://dmoz.org/.

[118] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.

[119] N. Paton, editor. *Active Database Features in SQL3 in Active rules in Database systems*. Springer Verlag, 1999.

[120] Information on clusters of pcs. http://www.alinka.com/fr/index.htm.

[121] T. Phelps and R. Wilensky. Robust hyperlinks cost just five words each. Technical report, UC Berkeley Computer Science, 2000. Technical Report UCB//CSD-00-1091.

[122] P. Pirolli, J. Pitkow, and R. Rao. Silk from a sow's ear: Extracting usable structures from the web. In *Proc. ACM Conf. Human Factors in Computing Systems, CHI*. ACM Press, 1996.

[123] M. Preda. Data acquisition for an xml warehouse. *DEA thesis Paris 7 University*, 2000.

[124] J. Pustejovsky, B. Boguraev, M. Verhagen, P. Buitelaar, and M. Johnston. Semantic indexing and typed hyperlinking. In *Proceedings of the American Association for Artifical Intelligence Conference, Spring Symposium, NLP for WWW*, 1997.

[125] S. Raghavan and H. Garcia-Molina. Crawling the hidden web. In *International Conference on Very Large Databases*, 2001.

[126] M. Rasmussen. On the use of condition monitoring for maintenance optimisation. In *ESReDA*, 1992.

[127] P. Resnik. Using information content to evaluate semantic similarity in a taxonomy. *IJCAI*, (95):448–453, 1995.

[128] P. Resnik. Semantic similarity in a taxonomy: An information-based measure and its application to problems of ambiguity in natural language. *Journal of Artificial Intelligence Research*, 1999.

[129] R. Richardson, A. Smeaton, and J. Murphy. Using wordnet as a knowledge base for measuring semantic similarity between words. In *Conference on Artificial Intelligence and Cognitive Sciense*, 1994. Dublin.

[130] C.J. Van Rijsbergen. *Information Retrieval, 2nd edition.* Dept. of Computer Science, University of Glasgow, 1979.

[131] G. Salton and M. McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill, 1983.

[132] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423 and 623–656, 1948.

[133] H. Small. Co-citation in the scientific literature: A new measure of the relationship between two documents. *Journal of the American Society of Information Science*, (24):265–269, 1973.

[134] R.T. Snodgrass, editor. *The TSQL2 temporal query language.* Kluwer Press, 1995.

[135] R.R. Sokal and P.H. Sneath. *Principles of numerical taxonomy.* W. H. Freeman, 1963.

[136] M. Steinbach, G. Karypis, and V. Kumar. A comparison of document clustering techniques. In *SIGKDD Workshop on Text Mining*, 2000.

[137] B. Stroustrup. *The C++ programming language.* Addison-Wesley, Reading, Massachusetts, special edition, 2000.

[138] I. Tatarinov, Z.G. Ives, A.Y. Halevy, and D.S. Weld. Updating XML. In *Proceedings of the ACM-SIGMOD*, 2001.

[139] Teoma search engine. http://www.teoma.com/.

[140] E. Terzi, A. Vakali, and M.S. Hacid. Knowledge representation, ontologies, and the semantic web. In *Web Technologies and Applications: 5th Asia-Pacific Web Conference*, 2003.

[141] THESUS. Web Application. `www.db-net.aueb.gr/thesus/`.

[142] A.A. Vaisman. OLAP, Data Warehousing, and Materialized Views: a Survey.

[143] I. Varlamis and M. Vazirgiannis. Web document searching using enhanced hyperlink semantics based on XML. In *IDEAS conference*, 2001.

[144] I. Varlamis, M. Vazirgiannis, M. Halkidi, and B. Nguyen. Thesus, a closer view on web content management enhanced with link semantics. *Transactions on Knowledge and Data Engineering*, To appear.

[145] Vivisimo search engine. http://www.vivisimo.com/.

[146] W3. Resource Description Framework (RDF). `www.w3.org/RDF`.

[147] W3. Simple Object Access Protocol (SOAP). `www.w3.org/TR/SOAP`.

[148] W3. Web Service Description Language (WSDL). `www.w3.org/TR/wsdl`.

[149] W3. XML Path Language (XPath). `www.w3.org/TR/xpath`.

[150] W3. XQuery 1.0 an XML query language. `http://www.w3.org/TR/xquery/`.

[151] W3C. *eXtensible Markup Language (XML) 1.0*, february 1998.

[152] World Wide Web consortium page on XML. http://www.w3c.org/TR/REC-XML.

[153] Y. Wang and M. Kitsuregawa. Link based clustering of web search results. In *WAIM conference*, 2001.

[154] WebRatio. A Modeling Language for Designing Web Sites (WebML). `www.webml.org`.

[155] J. Widom. Research problems in data warehousing. In *International Conference on Information and Knowledge Management (CIKM)*, 1995.

[156] J. Widom and S. Ceri. *Active database systems: Triggers and rules for advanced prcessiong.* Morgan-Kaufmann, California, 1995.

[157] Jennifer Widom. The starburst active database rule system. *Knowledge and Data Engineering*, 8(4):583–595, 1996.

[158] P. Willet. Recent trends in hierarchical clustering: a critical review. *Information Processing and Management*, 24:577–597, 1988.

[159] Wordnet web site. http://www.cogsci.princeton.edu/ wn/.

[160] Z. Wu and M. Palmer. Verb semantics and lexical selection. In *Annual Meetings of the Associations for Computational Linguistics*, number 32, pages 133–138, 1994.

[161] XML-DB. XUpdate - XML Update Language. `http://www.xmldb.org/xupdate/` .

[162] Xyleme s.a. http://www.xyleme.com/.

[163] Lucie Xyleme. A Dynamic Warehouse for XML Data of the Web. *IEEE - Data Engineering Bulletin*, 24(2):40–47, June 2001.

[164] Yahoo! http://www.yahoo.com/.

[165] O. Zamir and O. Etzioni. Web document clustering: a feasibility demonstration. In *Proceedings of the ACM-SIGIR*, 1998.