Selective Dissemination of Information in P2P Systems: Data Models, Query Languages, Algorithms and Computational Complexity

Christos Tryfonopoulos and Manolis Koubarakis

Intelligent Systems Laboratory
Dept. of Electronic and Computer Engineering
Technical University of Crete
73100 Chania, Crete, Greece
{trifon,manolis}@intelligence.tuc.gr
http://www.intelligence.tuc.gr

Technical Report TR-ISL-02-2003

Abstract

Much information of interest to humans is today available on the Web. People can easily gain access to information but at the same time, they have to cope with the problem of information overload. Consequently they rely on specialised tools and systems designed for searching, querying and retrieving information from the Web. It is however extremely difficult to stay informed without sifting through enormous amounts of incoming information, and without utilising tools and techniques that would capture the dynamic nature of the Web. The selective dissemination of information is a mechanism that can help users cope with this problem. In a selective information dissemination scenario, users post profiles or long-standing queries, which describe their information needs to some dedicated middleware. This middleware receives incoming information, decides whether it matches stored user profiles and delivers it to the interested subscribers.

In this work we deal with the problem of textual information dissemination in the context of distributed peer-to-peer systems. We put our main focus on data models and query languages especially designed for information dissemination, when information is in the form of text. We incrementally develop three such models and query languages moving from very simple ones to ones with more expressive power. We also identify problems that frequently arise in such a scenario and study their computational complexity. Finally we design, implement and experimentally evaluate two algorithms for the problem of filtering as it arises in textual information dissemination.

Chapter 1

Introduction

1.1 Introduction

Much information of interest to humans is today available on the Web. People can easily gain access to information but at the same time, they have to cope with the problem of information overload. Consequently they rely on specialised tools and systems designed for searching, querying and retrieving information from the Web. It is however extremely difficult to stay informed without sifting through enormous amounts of incoming information, and without utilising tools and techniques that would capture the dynamic nature of the Web. The *selective dissemination of information* is a mechanism that can help users cope with this problem. In a selective information dissemination scenario, users post *profiles* or *long-standing queries*, which describe their *information needs* to some dedicated *middleware*. This middleware receives incoming information, decides whether it matches stored user profiles and delivers it to the interested subscribers.

1.1.1 Overview

Selective dissemination of information to interested users is a problem that has recently received the attention of various research communities including researchers from agent systems [46, 80, 39, 126, 128], databases [51, 6, 103, 42], digital libraries [44], distributed computing [20, 19] and others.

We envision an information dissemination scenario in the context of a *distributed peer-to-peer (P2P) agent architecture* like the one shown in Figure 1.1. Users utilize their *end-agents* to post *profiles* or *documents* (expressed in some appropriate language) to some *middle-agents*[1]. End-agents play a dual role: they can be information producers and information consumers at the same time. The P2P network of middle-agents is the "glue" that makes sure that published doc-

[1]In this report we will use the terms *query* and *profile* interchangeably. In an information dissemination setting, a profile is simply a long-standing query.

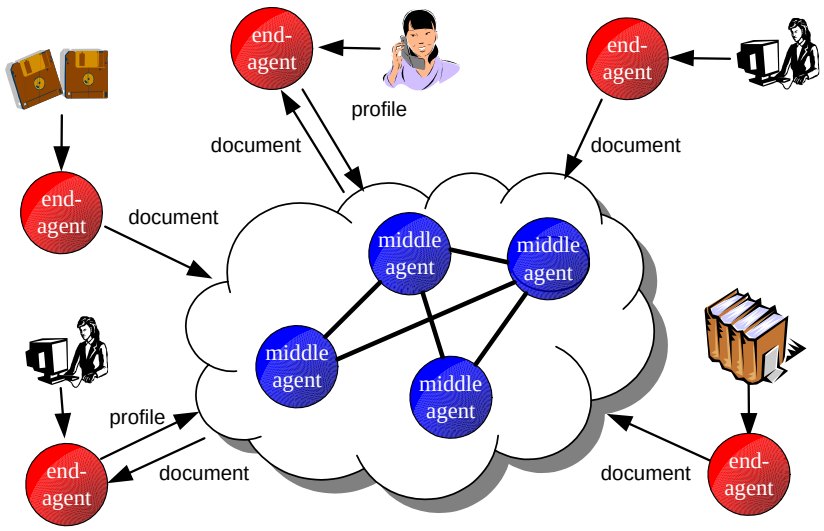

Figure 1.1: A distributed P2P agent architecture for information dissemination

uments arrive at interested subscribers. To achieve this, middle-agents forward posted profiles to other middle-agents using an appropriate P2P protocol. In this way, matching of a profile with a document can take place at a middle-agent that is as close as possible to the origin of the incoming document. Profile forwarding can be done in a sophisticated way to minimize network traffic e.g., no profiles that are less general than the one that has already been processed are actually forwarded. Moreover sophisticated techniques like summarization of profiles [132] can be utilized to further reduce the amount of data that is passed through the network.

In their capacity as information producers, end-agents can also post *advertisements* that describe in a "concise" way the documents that will be produced by them. These advertisements can also be forwarded in the P2P network of middle-agents to *block* the forwarding of *irrelevant* profiles towards a source. Advertisement forwarding can also be done in a sophisticated way using ideas similar to the ones for profile forwarding.

Most of the concepts of the architecture sketched above are explicit (or sometimes implicit) in the KQML literature and subsequent research on agent middleware based on KQML [46, 80, 39, 126, 128]. Unfortunately the emphasis in most of these systems is on a single central middle-agent, making the issues that would arise in a distributed setting difficult to appreciate. In our opinion, the best presentation of these concepts available in the literature can be found in [20] where the distributed event dissemination system SIENA is presented. Although SIENA does not use terminology from the area of agent systems the connection is obvious. In [74, 41, 72] it is shown how the main principles from SIENA can guide the development of a distributed agent architecture appropriate for information

dissemination.

This report concentrates on the problem of information dissemination assuming that we are only dealing with *textual* information. We are motivated by a desire to develop useful agent systems in a *principled* and *formal* way, and make the following technical contributions.

- We define formally the models $\mathcal{WP}, \mathcal{AWP}$ and $\mathcal{AWPS}$, and their corresponding languages for textual information dissemination in distributed agent systems. Data model $\mathcal{WP}$ is based on free text and its query language is based on the *boolean model with proximity operators*. The concepts of $\mathcal{WP}$ extend the traditional concept of proximity in IR [10, 27, 28] in a significant way and utilize it in a content language targeted at information dissemination applications. Data model $\mathcal{AWP}$ is based on *attributes* or *fields* with finite-length strings as values. Its query language is an extension of the query language of data model $\mathcal{WP}$. Our work on $\mathcal{AWP}$ complements recent proposals for querying textual information in distributed event-based systems [20, 19] by using linguistically motivated concepts such as *word* and not arbitrary strings. This makes $\mathcal{AWP}$ potentially very useful in some applications (e.g., alert systems for digital libraries or other commercial systems such as news dissemination, where similar models are supported already for retrieval). Finally, the model $\mathcal{AWPS}$ extends $\mathcal{AWP}$ by introducing a "similarity" operator in the style of modern IR, based on the vector space model [10]. The novelty of our work in this area is the move to query languages much more expressive than the one used in the information dissemination system SIFT [145] where documents and queries are represented by free text. The similarity concept of $\mathcal{AWPS}$ is an extension of the similarity concept pioneered by the system WHIRL [34] and recently also used in the XML query language ELIXIR [30]. We note that both WHIRL and ELIXIR target information retrieval and integration applications, and pay no attention to information dissemination and the concepts/functionality needed in such applications.

- We study the computational complexity of four problems that are fundamental in information dissemination systems following the architecture of Figure 1.1. These problems are:

  1. *Satisfiability.* Deciding whether a given query ϕ can be satisfied by any document at all. This functionality is necessary at each middle agent.
  2. *Satisfaction or matching.* Deciding whether an incoming document satisfies (or matches) a query ϕ.
  3. *Filtering.* Given a database of profiles db and an incoming document d, find all profiles $q \in db$ that match d. This functionality is very

crucial at each middle agent and it is based on the availability of algorithms for the satisfaction problem. We expect deployed information dissemination systems to handle hundreds of thousands or millions of profiles.

4. *Entailment or subsumption.* Deciding whether a profile is more or less "general" than another. This functionality is crucial if we want to minimize profile forwarding as sketched above.

- We concentrate on the filtering problem and propose two efficient main memory algorithms that can handle conjunctions of atomic queries in the model $\mathcal{AWP}$, and evaluate them experimentally. Our filtering algorithms are based on the algorithms developed in [144] and recently reimplemented and evaluated in [74, 41, 71]. Additionally, our experimental evaluation follows the proposal of [74] where the evaluation of information dissemination algorithms using a real document corpus and a set of realistic user profiles was originally presented.

This work was supported in part by project DIET (IST-1999-10088) funded by the IST Programme of the European Commission, under the FET Proactive Initiative on "Universal Information Ecosystems".

1.1.2 Organisation of the Report

This report is organised as follows. In Chapter 2 we survey recent research in agent middleware, which is the general area of research this report falls in. In Chapter 3 we define the data models and query languages specially designed for agent-based textual information dissemination. In Chapter 4 we formulate and study the computational complexity of four fundamental problems in information dissemination: the problems of satisfiability, satisfaction, filtering and entailment. In Chapter 5 we present and evaluate experimentally two efficient algorithms for the filtering problem in a textual information dissemination scenario. Finally in Chapter 6 we present our conclusions and propose directions for future research.

Chapter 2

Related Work

The exponential growth in the number of available information sources and services on the web, and in the number of users and specialized software attempting to take advantage of them, has spurred research on the design and implementation of middleware for open information environments. This chapter is a survey of recent research on middleware developed in the area of multi-agent systems with an emphasis on information management applications. We trace the origins of agent middleware in the concepts of mediator and facilitator, and review related work up to today. We also discuss some popular Internet systems (search engines, publish/subscribe systems and peer-to-peer systems) and show how they can be casted as multi-agent systems with a central role for agent middleware.

2.1 Introduction

The exponential growth in the number of available information sources and services on the web, and in the number of users and specialized software attempting to take advantage of them, has spurred research on the design and implementation of middleware for open information environments. The role of this middleware is twofold: to assist information requesters to find appropriate information providers and to provide additional value-adding services to both parties. Value-adding services can range from purely informational ones (such as translating, integrating, summarizing or abstracting information coming from many sources) to economic ones (such as serving as a trusted third-party for e-commerce transactions). This survey targets mainly information management applications thus economic issues will only briefly be mentioned.

Without help from specialized middleware, users are left with no other choice but to utilize any of the existing search engines (e.g., Google[1]), web directories

[1] http://www.google.com

(e.g., Yahoo[2]) or vertical portals[3]. Users are then faced with the task of manually browsing or querying various information sources, collecting relevant information and synthesizing it into a useful form.

To alleviate this problem various kinds of agents have recently been proposed in the context of open multi-agent systems: mediators [129], facilitators [54], matchmakers [78] and so on. This chapter is a comparative survey of research into these kinds of agent middleware. Related research on similar kinds of middleware has been carried out in several other areas of computer science:

- Databases and information systems, where the emphasis has been on middleware that integrates information collected from heterogeneous distributed sources [129, 138, 53], or middleware that collects published information and disseminates it to users with matching profiles [6, 145].

- Distributed systems and networks, where the role of middleware has been prominent in client-server architectures [134], resource discovery systems [14, 88], toolkits for building information gathering systems [15], web proxies [131] and wide-area event dissemination systems [21].

- Software engineering, where the role of middleware has been to assist in the discovery of software components, software interoperation, and development of large-scale systems through integration and reuse of existing components [58, 11].

In our tour of research on agent middleware, we sometimes find it appropriate to discuss and evaluate selected contributions coming from the above areas.

Before we start our presentation, let us carefully introduce the terminology to be used in the rest of this chapter. As we discuss in Section 2.2, previous surveys on agent middleware often use the same name to describe concepts that sometimes differ substantially. To avoid such confusion, the rest of this survey uses the generic terms end-agents and middle-agents, originally introduced in [39, 142], to describe the two different classes of agents that we will study. End-agents are agents that need or offer services. Middle-agents are agents that exist with the sole purpose of enabling interactions among end-agents [39, 142]. It is of course possible to have no middle-agents at all; we will also discuss this possibility in detail.

The needs or preferences of an end-agent become known in the form of requests that are expressed in some language and are communicated to other agents using an appropriate communication protocol. The services offered by end-agents are usually referred to as the capabilities of these agents. Capabilities are also expressed in some language and are communicated to other agents using an appropriate communication protocol.

[2]http://www.yahoo.com
[3]http://verticalnet.com

Previous research on agent middleware can be evaluated properly if one keeps in mind the following dimensions[4]:

- Communication i.e., what are the communication primitives and protocols used by end-agents and middle-agents? What are the languages used to express capabilities and requests? How can agents share their knowledge about a domain of expertise?

- Matching i.e., what are the algorithms used to match requests with capabilities? What is the computational complexity of these algorithms?

- Degree of involvement by middle-agents i.e., do middle-agents simply enable requesters to get in touch with providers, or do they intermediate transactions as well?

- Architecture i.e., is it assumed that there is a central middle-agent, many cooperating middle-agents, or no middle-agents at all? What are the advantages and disadvantages for each different alternative?

The questions that characterize each of the above dimensions will dominate much of the discussion in this survey[5]. But because agent researchers would like to see their agents being deployed on the Internet and used in the context of real applications, there are some more practical issues that deserve to be addressed in detail[6]:

- Scalability: agent systems deployed on the Internet must be able to scale to huge numbers of advertisements of capabilities and requests of service. Can existing agent technology face up to this challenge? What are the relevant technical problems that have been solved and what remains to be done?

- Robustness and fault-tolerance: the performance of agent systems deployed on the Internet must degrade gracefully in case that one or more components fail.

- Adaptivity: agent systems deployed on the Internet must be able to adapt gracefully to varying workloads and varying numbers of requesters and providers.

[4]Some of these dimensions were first presented by Chi Wong and Sycara in [142]. Katia Sycara and her colleagues have recently done a lot to clarify the issues involved in designing and implementing agent middleware. In this respect the present survey owes a lot to papers such as [39, 127, 142].

[5]The interested reader can also consult [142] for a different detailed list of questions used to create a taxonomy of existing middle-agents.

[6]Admittedly, there are other issues as well: security, trustworthiness etc. We will not spend any time on these issues in this survey since we only concentrate on agent middleware for information management applications.

- Privacy: agent systems must enable their owners to give out only as much information about themselves as it is necessary for carrying out their transactions.

- Web-friendliness: if agents are to be deployed on the Internet, it might be much easier if, right from the start, they are designed taking current Internet technology into account (relevant technologies here include these developed around W3C languages[7] XML, RDF etc.).

The rest of this chapter is organized as follows. Section 2.2 presents some of the early work that has paved the way for todays research on middle-agents. Section 2.3 is a general introduction to agent communication based on the languages KQML and FIPA-ACL while section 2.4 concentrates on the important role of middleware in the development of KQML and FIPA-ACL. Section 2.5 discusses existing implemented middle-agent systems, and illustrates the advantages and disadvantages of the choices made by their implementers. Section 2.6 concentrates on architecture alternatives for agent middleware. Section 2.7 analyzes the concept of privacy and how it relates to various kinds of middle-agents. Section 2.8 concentrates on performance issues. Finally, Section 2.9 presents our conclusions and speculates on avenues for future research.

2.2 Mediators and Facilitators

To the best of our knowledge, the earliest well-known work to discuss issues related to agent middleware is [37]. In that paper the so-called connection problem of how to find other agents that have the capabilities you need is discussed in the context of distributed problem solving.

Another early influential paper is [129] where the concept of mediation is presented. A mediator is a software module that exploits encoded knowledge about some heterogeneous sets of data sources to synthesize information for a higher layer of applications. According to [139], the tasks to be achieved by a mediator module are the following:

- Locating, accessing and retrieving relevant data from multiple-heterogeneous sources.

- Abstracting and transforming the retrieved data to a common model and level so they can be integrated.

- Integrating the transformed data according to matching keys.

- Reducing the integrated data by abstraction so that the information density of the data transmitted to decision-making applications increases.

[7] Web pages of the World Wide Web Consortium: http://www.w3c.org

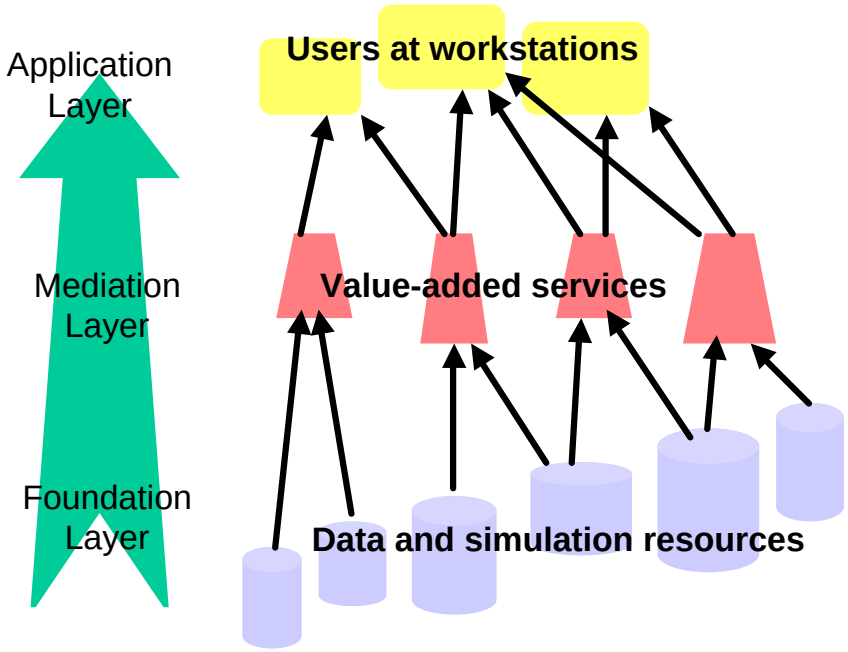

Figure 2.1: Transforming data into information using mediators

Thus the processing by a mediator creates an information chain where value is generated by transforming data into information (see Figure 2.1 which is taken from [140]).

The concept of mediator as introduced by Wiederhold [129] is very general and informal. More recent work on mediators has followed the original spirit of [129] but has concentrated mostly on data integration. Examples of projects in this area include the Information Manifold [87], SIMS [8], ARIADNE [68], TSIMMIS [53], Infomaster [56], MOMIS [13] and others. All these projects consider multiple heterogeneous data sources (e.g., web pages written in HTML, relational databases, object-oriented databases, legacy systems etc.) and integrate them using some common data model. Queries combining information from many sources are then handled in the integrated representation.

The development of agent middleware has also been of paramount importance from early on in multi-agent systems research [54, 46]. Starting with the paper of Genesereth and Ketchpel on software agents [54], we see the introduction of federated agent systems and facilitators. The vision behind the concept of software agent in [54] is enabling arbitrary software or hardware systems to interoperate by sharing knowledge represented in the declarative knowledge representation language KIF[8] (an acronym for Knowledge Interchange Format technically an extension of first-order logic). Figure 2.2 (taken from [54, 55]) shows an example of a federated agent system with three communities of agents. In this architec-

[8]http://logic.stanford.edu/kif/kif.html

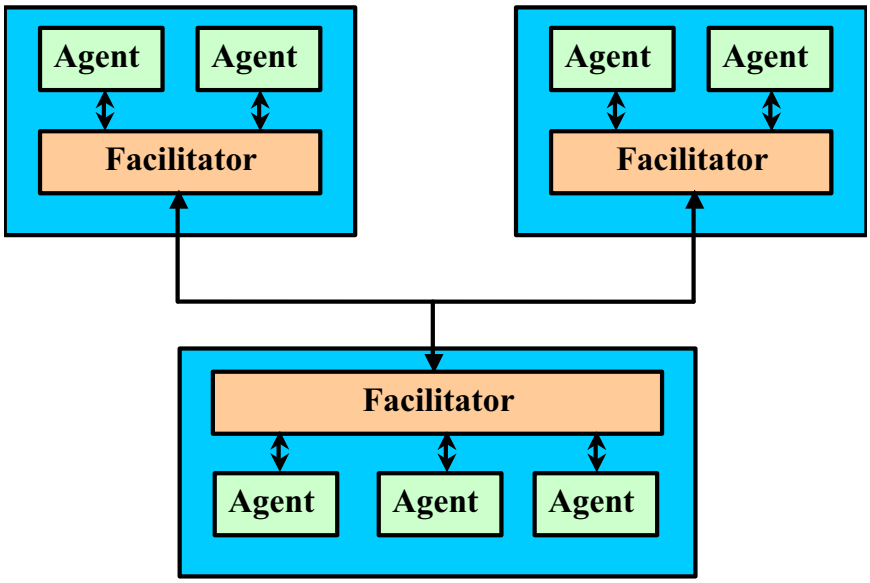

Figure 2.2: A federated agent system.

ture, individual agents can communicate directly only with their local facilitators and facilitators communicate with one another. In this architecture there can be many agent communities with their facilitators, residing on one or many machines, and the connections between the facilitators can be arbitrary.

Organizing agents in a federation (such as in Figure 2.2) is, of course, only one of the many possible architectural alternatives for a multi-agent system. Various other alternatives are discussed in Sections 2.5 and 2.6.

The concept of facilitator, as envisioned in [54], is a very general one. According to [121], facilitators can provide many services including:

- White pages i.e., finding information about agents given their symbolic name. For example, What is the internet address of agent A?
- Yellow pages i.e., finding the identity of agents with certain capabilities. For example, What agents are capable of answering queries about the weather in Greece?
- Direct communication i.e., sending a message to another agent.
- Other services such as problem decomposition (for problem solving agents), translation services etc.

Although the functionality envisioned for a facilitator in [54, 121] is rather complicated and can lead to inflexibility (e.g., all communication has to go

through the facilitator), facilitators have been implemented and used successfully in general agent architectures e.g., OAA [92]. Other researchers have chosen to breakdown the functionality of a facilitator, and this has given rise to closely related but simpler concepts such as communication facilitators [47] and various types of matchmakers [78] (all these concepts are surveyed below).

It is important to point out here that the terms mediator, facilitator, matchmaker and broker can easily confuse a newcomer to this area since different authors often use the same concept to express different functionality. This is particularly obvious when authors compare their favourite concept to a concept used by another researcher. For example, [54, p. 52] claims that the concept of a facilitator derives from and generalizes the concept of a mediator, [139, p. 36] clearly implies that the concept of a mediator subsumes the concept of a facilitator, while [140, p. 52] which adopts a more conciliatory approach, stresses the different visions behind each concept, and illustrates their comparative advantages.

We hope the present survey chapter will help to clarify the issues involved in designing agent middleware and will serve as a good starting point for newcomers to this area. But before we get into the main part of our survey, let us briefly discuss a basic issue: how do end-agents and middle-agents communicate?

2.3 An Introduction to Agent Communication

The ability to communicate has always been considered fundamental for the development of human and other animal societies. Naturally, early research on intelligent agents has taken up the topic of agent communication very seriously. In fact, some researchers like Michael Genesereth have gone so far as to identify agency with the ability to communicate using an agent communication language [54].

An agent communication language (ACL) is a means of exchanging information and knowledge among agents [83]. ACLs are clearly distinguished from traditional ways of information exchange among interoperating applications such as remote procedure calls, remote method invocations or object request brokers. Contrary to traditional approaches, ACLs emphasise that communication requires the exchange of declarative sentences with carefully defined semantics, not just simple syntactic objects. In the rest of this section we briefly present the two most popular ACLs to-date: the Knowledge Query and Manipulation Language (KQML) [46] and the ACL defined by the Foundation for Intelligent Physical Agents (FIPA[9]).

[9]FIPA web pages: http://www.fipa.org

2.3.1 KQML

KQML was first introduced as one of the results of the Knowledge Sharing Effort (KSE [10]) , which has influenced current efforts in inter-agent communication approaches.

The KSE was initiated as a research effort circa 1990 with encouragement and relatively modest funding from U.S. government agencies (DARPA especially). The KSE was highly active for roughly five years thereafter, and enjoyed the participation of dozens of researchers from both academia and industry; the researchers represented various branches of the AI community. Its goal was to develop techniques, methodologies and software tools for knowledge sharing and knowledge reuse between knowledge-based (software) systems, at design, implementation, or execution time. Agents, especially intelligent agents, are an important kind of such knowledge-based systems (other kinds include expert systems or databases, for example). The central concept of the KSE was that knowledge sharing requires communication, which in turn, requires a common language; the KSE focused on defining that common language

In the KSE model, agents (or, more generally, knowledge-based systems) are viewed as (virtual) knowledge bases that exchange propositions using a language that expresses various propositional attitudes. Propositional attitudes are three-part relationships between

- an agent,
- a content-bearing proposition (for example, *it is raining*), and
- a finite set of propositional attitudes an agent might have with respect to the proposition (for example, believing, asserting, fearing, wondering, hoping, and so on).

For example, $< a, fear, raining_{now} >$ is a propositional attitude.

The KSE model includes three layers of representation: (1) specifying propositional attitudes; (2) specifying propositions (i.e., *knowledge*) - this is often called the (propositional) content layer; and (3) specifying the ontology [60](i.e., vocabulary) of those propositions. The KSE accordingly includes a component (with associated language) for each of these: Knowledge Query and Manipulation Language (KQML) for propositional attitudes, Knowledge Interchange Format (KIF [11]) for propositions, and Ontolingua.

Within the KSE approach, the three representational layers are viewed as mainly independent of another. In particular, the language for propositional content (i.e., the content language) can be chosen independently from the language for propositional attitudes. In other words, in the KSE approach, the

[10] http://www.cs.umbc.edu/kse/

[11] http://logic.stanford.edu/kif/ and http://www.cs.umbc.edu/kif/

```
(ask-one
  :sender joe
  :content (PRICE IBM ?price)
  :receiver stock-server
  :reply-with ibm-stock
  :language LPROLOG
  :ontology NYSE-TICKS)

(tell
  :sender stock-server
  :content (PRICE IBM 14)
  :receiver joe
  :in-reply-to ibm-stock
  :language LPROLOG
  :ontology NYSE-TICKS)
```

Figure 2.3: Two examples of KQML.

role of an ACL, namely KQML's in the case of the KSE (or FIPA ACL's, much later) is only to capture propositional attitudes, regardless of how propositions are expressed, even though propositions are what agents are "talking" about [12].

KQML is a high-level communication language based on a collection of performatives that are used to express a propositional attitude (involving two agents, e.g., agents Joe and Stock-Server) about a content-bearing proposition (e.g., the price of IBMs stock is $14). As an example, Figure 2.3 shows a request of agent Joe about IBM's stock price as it would be expressed in KQML, and the reply of the agent Stock-Server [47].

The designers of KQML have adopted a Lisp-like syntax, which seems to have served them well throughout the development of the language [83]. KQML expressions like the ones in Figure 2.3 are messages that can be understood in terms of three layers: the content layer, the communication layer and the message layer. The content layer expresses a proposition which is included in the slot :content of a KQML expression. KQML makes no commitment with respect to the language these propositions are written and various content languages have been used in applications (see Section 2.4 below). In the examples of Figure 2.3 the content language is KIF. The communication layer captures a set of communication parameters such as the identity of the sender and receiver and a unique identifier associated with the communication (slots :sender, :receiver and :reply-with in Figure 2.3).

Finally, the message layer identifies a performative such as ask-one or tell. The

[12] In a similar spirit, the approach of the technical committee that worked on FIPA ACL is that the content language should be viewed as orthogonal to the rest of the ACL message type.

message itself does not identify the network protocol with which the message is delivered (e.g., TCP/IP, IIOP or other); in fact, any of implemented (by the involved agents) transport protocols may be used and it is typically expected that the available protocols have been made known prior to the communication between the agents. Performatives express an attitude regarding the proposition in the slot :content of the message. The term performative or speech act has its origins in the philosophy of language [9, 113] where it is used for verbs can be uttered so that they perform some action. Performatives in KQML correspond to message types in traditional communication protocols research, but of course performatives have a much richer semantic theory associated with them.

Since KQML makes no commitment regarding the content language used in messages, the message layer includes some more concepts to allow the successful analysis and delivery of messages in KQML implementations. For example, slot :language declares the content language used in the :content field, while :ontology declares the ontology that needs to be consulted if the receiving agent wishes to understand the content of the message [13].

Depending on the application, many content languages have been used in multi-agent systems utilizing KQML. These include free-text [78], KIF, versions of Prolog and other logic programming languages [59] and languages for structured objects such as MAX [78]. Recently, there has been a move towards encoding KQML or FIPA-ACL messages in XML so that the development/maintenance of ACL parsers is simplified and integration with other systems developed for the WWW becomes easier. We expect this trend to continue in the near future particularly due to the recent efforts to develop languages, tools and techniques to make more content on the Web machine understandable (this is the recent Semantic Web effort in the U.S. and Europe [45, 125]) [14].

Semantics

When they were originally introduced, KQML performatives were not given a clear semantics but a substantial amount of work has been done since then to alleviate this problem [33, 82]. Cohen and Levesque were the first to point out these semantic problems with KQML and introduced a theory of speech acts (based on a theory of rational action) which can be used to provide a formal semantics for KQML performatives. Work on the semantics of KQML along similar lines has also been done by Labrou and Finin [81, 82]. In both of these approaches, KQML performatives are understood as speech acts that alter the mental state of the agents participating in a conversation. Mental states of agents are modelled in terms of primitives such as beliefs, desires, goals, etc. along the

[13] The KSE effort actually contributed a lot of new ideas and prototype systems dealing with ontologies e.g., Ontolingua.

[14] Naturally the semantic web efforts have also accelerated current research and development on ontologies (see OIL [135] and DAML [63]).

lines of BDI theory [109]. Related work in this area has also been done by Sadek [111] and Singh [119, 118]. Recently, Singh [120] and Pitt and Mamdani [106, 107] have criticized this emphasis on intentional approaches as being too strong for heterogeneous agent applications. The original KQML specification suggested an implicit sequencing of messages in agent interactions. First in [84] and later in [82, 36] the idea of conversations for communicating agents that use an ACL, was introduced and further explored. Conversations mark a shift from individual messages to sequences (exchanges) that agents engage in order to perform certain tasks. The emphasis shifts from the agent's internals to the agent's behavioral patterns. Other researchers used the concepts of agent conversations as a basis for an ACL semantics. [106] suggest that a protocol-based approach that moves the emphasis from mental states of agents to what an agent does in response to receiving a message might be more appropriate in many applications. In [105] this approach is used to define multi-party agent conversations that can be used to define any of the middle-agent protocols to be discussed in Section 2.4 (also different protocols such as auctions).

KQML has been adopted as a communication language in many multi-agent systems targeted to specific applications (see Section 2.4 below). There are also various APIs that enable one to implement applications with agent communication capabilities (e.g., JAFMAS [29] and Jackal [36] are two of the most recent ones).

2.3.2 FIPA-ACL

The Foundation for Intelligent Physical Agents is an international consortium of universities and companies formed in 1996. FIPA is devoted to the standardization and promotion of languages, tools and architectures for the development of intelligent agents. FIPA operates by concentrating on annual specification deliverables. The most recent specification is FIPA 2000, whereas older specifications also exist (FIPA 97 and FIPA 98), but are now considered obsolete. All these specifications are publicly available[15].

FIPA-ACL is the agent communication language created by FIPA building on the international experience gained with KQML. FIPA-ACL adopts the Lisp-like syntax of KQML and the principle of separating the performative or communicative act from the proposition this communicative act applies. Figure 2.4 shows the example of Figure 2.3 in FIPA-ACL.

As the reader can see there are only superficial differences with KQML in the above example (e.g., the communicative acts inform and query-ref correspond to KQMLs tell and ask-one). The early specifications of FIPA-ACL were criticized for its lack of facilitation primitives such as broker, recommend and recruit that were extensively used by system developers used to KQML [83]. The general

[15]FIPA web pages: http://www.fipa.org

```
(query-ref
    :sender (agent-identifier :name joe)
    :receiver (set (agent-identifier :name stock-server))
    :content (price IBM ?price)
    :language FIPA-SL
    :ontology NYSE-TICKS
    :reply-with ibm-stock)

(inform
    :sender (agent-identifier :name stock-server)
    :receiver (set (agent-identifier :name joe))
    :content ``price(IBM,14)''
    :language Prolog
    :ontology NYSE-TICKS
    :in-reply-to ibm-stock)
```

Figure 2.4: Two examples of FIPA ACL

primitive proxy has been added to the latest FIPA-ACL specification to account for various KQML performatives (see Section 2.4).

The semantics of FIPA-ACL are defined in terms of SL, a quantified multi-modal logic with modal operators for beliefs, desires, uncertain beliefs and intentions [111]. The use of a formal language such as SL allows an elegant distinction between primitive communicative acts (inform, request, confirm, disconfirm) and non-primitive ones (i.e., those that can be defined in terms of primitive acts) as traditionally found in theories of action.

FIPA-ACL is currently gaining ground as the communication language of choice in multi-agent systems for various applications [99], gradually replacing KQML as the ACL of choice. Since KQML and FIPA ACL share the same foundational concepts the transition has been relatively smooth ([83] offers a detailed comparison between KQML and FIPA ACL).

2.4 Middle-agents, KQML and FIPA-ACL

One of the design criteria for agent communication languages has been to introduce appropriate communication primitives that would enable the creation of various interesting agent architectures based on middle-agents [47]. KQML contains a set of such performatives: recommend-one, recommend-all, recruit-one, recruit-all, broker-one, broker-all, subscribe and advertise [48, 47].

Obviously, the simplest kind of a multi-agent system is one where there is a

fixed number of agents (e.g., A and B) and each agent knows the internet address and capabilities of the other (Figure 2.5a). In this case, if agent A wants to know whether agent B believes that a proposition p is true, then using a simple point-to-point protocol, it can send a query about p to B. Upon receipt of the query, agent B will look-up its knowledge base and return with an appropriate reply.

The need for a middle-agent is apparent when the situation becomes more dynamic i.e., when agent A does not know what other agents are available or where they are located or what capabilities they might have. The introduction of a middle-agent M gives rise to some more interesting protocols for communication and information routing [78, 47]:

- The recommending protocol: In this case (see Figure 2.5b) agent A can request from a middle-agent M to recommend an agent that can process a performative e.g., ask-if(p). In KQML this can be done using the performative recommend-one (the meaning of recommend-all is analogous). If another agent B has already advertised to M that B is capable of processing performative ask-if(p) then M will reply to A informing him of Bs location in the network. Agent A is then free to communicate directly with B.

- The recruiting protocol: In this case (see Figure 2.5c) agent A can request a middle-agent M to recruit an agent that can process a performative e.g., ask-if(p). In KQML this can be done using the performative recruit-one (the meaning of recruit-all is analogous). The difference from the previous case is that, if another agent B has already advertised to M that it is capable of processing performative ask-if(p), then M will forward the query ask-if(p) to B and B will send a response back directly to A.

- The brokering protocol: In this case agent A can request a middle-agent M to find another agent which can process a given performative, to collect the reply and to forward the reply to A. This scenario is shown in Figure 2.5d and the performative broker-one is used.

- The publish/subscribe protocol [16]: In this case agent A can subscribe to receive updates each time there is a change in the answer to a performative e.g., ask-if(p). Thus in this scenario, agent M will inform A about the truth value of p immediately upon receiving the performative subscribe and, subsequently, every time the truth value of p changes.

Contrary to KQML, the communication language FIPA-ACL contains only one message type, proxy, which has been introduced to support various kinds of

[16] [78] call this case content-based routing. We avoid this terminology here because content-based routing is better thought of as addressing and routing technique which can be used profitably in applications such as event dissemination services [21] or intra-agent communication [122].

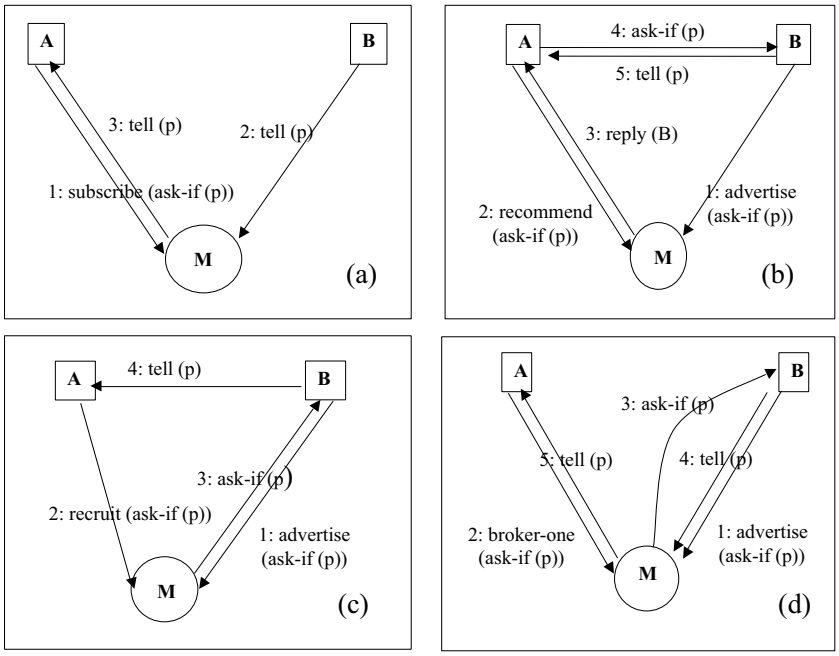

Figure 2.5: Various Protocols Involving Middle-agents

middle-agents[17]. The proxy message type is quite general and can be used to implement recommending, recruiting, brokering and publish/subscribe protocols as they have been presented above[18].

The middle-agents presented in the above paragraphs enable other agents to find appropriate information producers or consumers. The question that arises then is how producer and consumer agents find middle-agents in the first place. In implementations where a single (global or local) middle-agent exists, this is usually achieved by having the name and address of the agent fixed and known to other agents [47]. In a distributed setting (such as the ones discussed in Section 2.5) the name and address of middle-agents can be found using out-of-band means or by using some sort of hierarchical scheme like DNS.

2.5 Middle-Agents at Work

Let us now turn our attention to representative examples of middle-agents that have been implemented and used in the context of various agent-based systems. Our discussion will focus on the interaction protocols used by these middle-agents and the languages for representing capabilities and requests. In some cases we

[17]Foundation for Intelligent Physical Agents. FIPA Communicative Act Library Specification, 20/10/2000. Available from http://www.fipa.org/repository/cas.html

[18]Foundation for Intelligent Physical Agents. FIPA Interaction Protocols (IPs) specifications. Available from http://www.fipa.org/ips.html

```
(advertise
    :sender p
    :receiver mm
    :lang kqml
    :content    (ask-one
                    :lang kif
                    :content(subcomponent-of ?x ?y)))

(subscribe
    :sender c
    :receiver mm
    :lang kqml
    :content    (ask-about
                    :lang max
                    :content
                          [(trouble-report ?x)
                           (match ?x
                                [(subject  ".*gimbal.*" )])]))
```

Figure 2.6: Examples of KQML messages in SHADE

will also hint at some details of their matching algorithms. Towards the end of this section, we demonstrate the power of the middle-agent concept, by showing how the functionality of current popular Internet technologies can be understood as multi-agent systems with a central role for middle-agents (e.g., search engines, publish/subscribe systems and peer-to-peer systems). The relation between these technologies and multi-agent systems is central to Section 2.6 as well, where we concentrate on architectural issues.

2.5.1 The SHADE and COINS Middle-Agents

SHADE was one of the first projects to utilize a middle-agent based on the ideas on agent middleware developed by the KQML community [77]. The main purpose of the SHADE middle-agent was to assist designers and engineers in the processes of concurrent design and collaborative engineering. Versions of the SHADE middle-agent have also been used in other applications such as a satellite imagery clearinghouse [79].

The SHADE middle-agent uses KQML as its communication language and supports all the protocols presented in Section 2.4. There are two content languages that can be used. The first one is a subset of KIF, and the second one is a logic-based language for structured objects and string patterns called MAX. Figure 2.6 gives two examples of KQML messages utilizing KIF and MAX.

Matching advertisements to requests in the SHADE middle-agent is based solely on the content of the advertisements and requests i.e., it does not involve any background knowledge or ontology. No detailed algorithms for matching are given in [78] so one can draw no conclusions regarding the efficiency of the matching process.

COINS is an acronym for COmmon INterest Seeker, another middle-agent implemented by the developers of the SHADE middle-agent [78]. The special feature of COINS is that it uses a content language based on free-text (or equivalently a weighted vector of keywords). This is rather reasonable given the special nature of its target application, which is matching users with similar interests. The matching algorithms in COINS are based on well-known text similarity metrics (e.g., tf-idf) as pioneered by the SMART system [112].

2.5.2 Middle-Agents in RETSINA

RETSINA is a multi-agent system infrastructure developed at CMU by Katia Sycara and her colleagues [38, 126, 127]. RETSINA supports the construction and deployment of four basic agent types:

- interface agents that interact with users, receive input, and display results
- task agents that help users plan and carry out problem-solving activities
- information agents that provide access to heterogeneous information sources, and
- middle agents that help match agents that request services with agents that provide services.

The ACL used by RETSINA agents is KQML but the communication module has been designed in a modular fashion so that it can easily support other ACLs such as FIPA [116]. Middle agents in RETSINA can therefore use any of the interaction protocols discussed in Section 2.4.

One of the important contributions of work on middle-agents in RETSINA is the introduction of a new content language, called LARKS (Language for Advertisement and Request for Knowledge Sharing). LARKS is a very expressive language designed to facilitate the description of services provided by agents and being available on the Web.

A LARKS service specification is a frame with the following slot structure:

- Context, which describes the context where the service is applicable. The exact syntax for specifying contexts is not given in [126, 127] but one can assume from the discussion that it is a list of English words possibly representing concepts from a known ontology.

- Input, where the inputs to the service are declared (as variables)
- Output, where the outputs to the service are declared (as variables)
- Types, which declares the types of variables
- InConstraints, where constraints on the input variables are declared. These constraints can be expressed in some constraint language (e.g., Horn clauses augmented with arithmetic constraints in the spirit of constraint logic programming is used in [126, 127]).
- OutConstraints, where constraints on the ouput variables are declared.
- ConcDescriptions, where the meaning of the words used in other fields (e.g., in Context) is specified by relying on a given local domain ontology. Local ontologies can be formally defined using any terminological knowledge representation language and ITL is chosen in [126, 127].

An example of a service specification in LARKS is shown in Table 2.1. This service finds information on computers as it would be done e.g., by a search engine. Notice that both advertisements of capabilities and requests for services are expressed by a service specification in LARKS and are wrapped in an appropriate KQML message.

The inventors of LARKS have discussed a variety of ways two service specifications can match (exact match, plug-in match, relaxed match) and discuss matching algorithms applicable in each case [126, 127]. Although the discussion in [126, 127] is very interesting, the authors adopt a rather informal presentation of various issues so the reader is only left with the general impression that strict notions of matching are computationally expensive (NP-hard at least) while simpler ones are computational. What seems to be missing here is a careful definition of the syntax and semantics of a service definition, and then a detailed analysis of the computational complexity of matching for different classes of constraints and concept descriptions. It would also be interesting to point out what kinds of matching are appropriate in various configurations of middle-agents such as the ones discussed in Section 2.6 below.

The RETSINA middle-agents have been used in various applications including the management of a stock portfolio (the system WARREN described in [101]). One such middle-agent called A-Match is available on the Web [19] and can be exploited by users wishing to advertise agents or querying available agents for services. For a comparison of RETSINA with OAA another open infrastructure for developing multi-agent systems, the reader is invited to see [57].

The recent information integration framework XIB is based on the lessons learned in RETSINA but utilises languages and tools based on XML [90].

[19] www.cs.cmu.edu/ softagents/a-match/index.html.

Context	Computer * Computer
Types	InfoList = ListOf(model : Model*ComputerModel), brand: Brand*Brand price: Price*Money, color: Color*Colors);
Input	brands: SetOf Brand*Brand; areas: SetOf State; processor: SetOf CPU*CPU; priceLow*LowPrice: Integer; priceHigh*HighPrice: Integer;
Output	Info: InfoList;
InConstraints	
OutConstraints	sorted(Info)
ConcDescriptions	Computer = (and Product (exists has-processor CPU) (all has-memory Memory) (all is-model ComputerModel)); LowPrice = (and Price (ge 1800)(exists in-currency asset (USD))); HighPrice = (and Price (le 50000)(exists in-currency asset (USD))); ComputerModel = Asset(HP-Vectra, PowerPC-G3, Thinkpad770, Satellite315); CPU = asset(Pentium, K6, PentiumII, G3, Merced) [Product, Colors, Brand, Money]

Table 2.1: An example of a service specification in LARKS

2.5.3 Middle-Agents in Information Integration Systems

There are a few interesting middle-agents implemented as parts of research projects in information integration as discussed in Section 2.2 above. The one most closely related to the middle-agents presented above is the middle-agent of InfoSleuth, a project on the integration of heterogeneous information using a semantic approach based on ontologies [12, 22]. The InfoSleuth middle-agent receives and stores advertisements from other InfoSleuth agents regarding their capabilities. Based on these advertisements, it responds to queries from users as to what agents are able to satisfy their specific requests (i.e., it implements the recommending protocol discussed in Section 2.4). The InfoSleuth middle-agent uses KQML as its communication language. Details of the language for advertisements and requests can be found in [22].

Infomaster is another information integration system which uses a middle-agent [56] to access information over heterogeneous sources including Z39.50, SQL databases etc. The middle-agent in Infomaster receives a request from other Infomaster agents, consults its knowledge base about the capabilities of sources, and finally synthesizes and returns an answer to the requestor (i.e., it implements some version of the brokering protocol described in Section 2.4). There are various other research projects in information integration that rely on a middle-agent like

the one in Infomaster (e.g., see the mediators in Information Manifold [87], SIMS [6], ARIADNE [68], TSIMMIS [53] and MOMIS [13]).

2.5.4 Search Engines as Middle-Agents

In addition to the research prototypes discussed above, there are many current popular technologies that can be interpreted as multi-agent systems with a central role for middle-agents. The first example that comes to mind is the case of search engines (e.g., Google or AltaVista) interpreted as middle-agents [20]. For a search engine to work, crawlers collect information from web pages around the globe and index it by keywords using some sophisticated full-text indexing (e.g., see [17]). This can be thought of as passive advertising of information capabilities on the part of the owners of web sites. Active advertising is also possible by submitting web pages to search engines. Users submit keyword-based queries to search engines and get back ranked lists of URLs pointing to web pages containing these keywords [21].

2.5.5 Middle-Agents and Publish/Subscribe Systems

It is also interesting to compare the concept of a middle-agent as it was developed in multi-agent systems to the middleware that has been developed for publish/subscribe (or pub/sub) systems. The pub/sub paradigm became popular in the last decade and has resulted in the implementation of many commercial systems and research prototypes [66, 124, 21, 43]. In a pub/sub system we have information providers, information consumers and middleware. Information providers publish events to the system and information consumers subscribe to particular categories of events within the system. Finally, the middleware is responsible for routing published events to appropriate subscribers.

Pub/sub systems can be distinguished into the following categories depending on their subscription language [21]:

- Channel-based, where consumers subscribe to all notifications sent across an explicitly-designated channel (e.g., Java Message Service [124]).
- Subject-based, where consumers subscribe to all notifications that the publisher has identified as being relevant to a particular subject (e.g., ToolTalk [66]).

[20] The exact details of this interpretation does not matter; the interested reader can see the details of the COINS matchmaker discussed in Section 2.5.1 where the content language consists of free text or a weighted keyword vector [78].

[21] The exact indexing and ranking technology used in various search engines is usually confidential and only systems such as Google that originated in an academic environment have published details of their implementations [17, 31].

```
string exchange=NYSE string symbol=DIS float change > 0

string exchange=NYSE string symbol=DIS float prior=105.25 float
change=-4 float earn=2.04
```

Figure 2.7: A subscription and a notification in SIENA

- Content-based, where consumers subscribe to all notifications whose content matches consumer-specified predicates (e.g., SIENA [21], Gryphon [4], Le Subscribe [43]).

From the discussion of Section 2.4, it must be clear to the reader that content-based pub/sub systems are immediately realisable by agents implementing the publish/subscribe protocol. As an example, let us consider the pub/sub system SIENA [21] and see how it can be realised using ideas from multi-agent systems.

SIENA is an event notification service for use in wide-area networks such as the Internet [21]. It is implemented as a network of servers that offer access points to clients. Clients use the access points to advertise high-level descriptions of events that they generate and to publish notifications of such events. Clients can also use access points to subscribe for receiving notifications of interesting events. The power of SIENA lies in its expressive language for notifications, advertisements and subscriptions, and in its sophisticated algorithms for routing notifications of events to interested clients [21].

The language for notifications, advertisements and subscriptions in SIENA is based on a data model that supports an untyped set of typed attributes. Each attribute has a name, a type and certain associated operators (including equality). An example of a subscription and a matching notification in SIENA is given in Figure 2.7.

In a middle-agent like the one implemented in SHADE [78] a subscription and a notification like the one in Figure 2.7 can be realized using the publish/subscribe protocol of Section 2.4 as follows. An information consumer will send a subscription message to the middle-agent using the KQML performative subscribe, content language KIF, and content

(ask-if (and (stock-price-change-event NYSE DIS ?prior ?change ?earn) (> ?change 0)))).

For the matching notification of Figure 2.7 to become available, an information producer should send a message to the middle-agent using the KQML performative tell, content language KIF and content

(stock-price-change-event NYSE DIS 105.25 4 2.04).

Notice that we assume no local ontology and matching will be carried out solely by logical inference.

In a similar way, selective information dissemination systems such as SIFT [145] or XFilter [6] can also be casted as multi-agent systems.

2.5.6 Middle-Agents and Peer-to-Peer Systems

Peer-to-peer systems have recently received huge popularity especially due to the legal battles involving Napster[22]. It is interesting to view Napster as a multi-agent system involving end-agents (Napster clients) sharing resources with the help of middle-agents (Napster servers) [49]. Napster resources (i.e., MP3s) can be described by a simple ontology and communication can be done in KQML or FIPA-ACL. To be able to share MP3s with other agents, an end-agent registers with a middle-agent. Then, the end-agent advertises its resources to the middle-agent, and in response, the middle-agent subscribes to the end-agents resources so that it will be notified of any possible changes. When an end-agent wants to find an MP3, it asks the middle agent to recommend all end-agents that have advertised it. Later on, if it chooses to contact one of these end-agents to obtain the resource, it also advertises it with the middle-agent. It is an interesting challenge to build a version of Napster that is based totally on multi-agent technology and demonstrate the benefits of this approach. Other popular peer-to-peer systems such as Gnutella[23] and Freenet[24] can be viewed as multi-agent systems in a similar way. Because the architectural choices in these systems are important, we leave their discussion for the next section.

2.5.7 Middle-Agents and Web Services

In the last two years, we have experienced enormous momentum towards making available back-end functionality as "web services." Web services is a broad term used to describe self-contained, self-describing, modular applications that can be published, located, and invoked across the Web and can perform functions than can range from simple requests to complicated business processes. Web services represent a component of the next iteration of the web, often referred to as the *Semantic Web*; Tim Berners-Lee describes it as "an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation."

The shift towards web services has resulted to significant efforts and stan-

[22]Napster web site: http://www.napster.com

[23]Gnutella web site: http://www.gnutella.wego.com

[24]Freenet web site: http://freenet.sourceforge.net

dards such WSDL, [25] UDDI [26] and SOAP, [27]. Although existing UDDI servers are currently no more than directories for publishing web services with a very simple search interface for service discovery, the ultimate goal is to automate the discovery and use (through SOAP and WSDL) of broadly available web services. Eventually, UDDI servers will function as brokers that match producers and consumers of services, perhaps offering a more fine-grained and semantically powerful discovery capabilities (current UDDI servers only provide the simplest of text-based search). Furthermore, there is a potential for another class of brokers, either enhanced UDDI servers, or applications (agents) that can access UDDI servers, which can offer enhanced services through the composition and/or aggregation of individual services published through UDDI servers.

The stack of technologies for creating service brokers for the purposes of (1) matching providers of services with consumers of services, and (2) creating new value-adding services through aggregation of simpler services includes existing industrial efforts (UDDI, WSDL and SOAP) that provide match of the necessary "plumbing" for connecting to and accessing such services but automated discovery and invocation of web services will also require a "deeper" semantics of web services input, output and requirements (policies). Semantic web technologies can hopefully provide some of the latter,*i.e.*, a Web Ontology Language like DAML or DAML+OIL [28], for semantically enhanced service descriptions and languages for describing properties and capabilities of services, such as DAML-S [29].

2.6 Architectural Alternatives for Agent Middleware

The previous section presented a number of implemented multi-agent systems that utilise various kinds of middle-agents, and also some popular Internet technologies (e.g., pub/sub systems) that can be casted as multi-agent systems with a

[25]Web Services Description Language (http://www.w3.org/TR/wsdl) is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint.

[26]Universal Description, Discovery, and Integration (http://www.uddi.org/) is a specification for describing (rather simply) services and discovering services. UDDI servers are used as directories of services, where providers can publish their services and "consumers" can search for them.

[27]Simple Object Access Protocol (http://www.w3.org/TR/SOAP/) is a lightweight, XML-based protocol for exchange of information that consists of: an envelope that defines a framework for describing what is in a message and how to process it, encoding rules for expressing instances of application-defined datatypes, and a convention for representing remote procedure calls and responses.

[28]http://www.daml.org

[29]http://www.daml.org/services/

central role for middleware. It is worthwhile to consider the architectural alternatives in the design of these systems and the choices made by their implementers. Of particular interest to us is the degree of distribution of the functionality of middle-agents in these systems. After all, previous research in distributed systems shows that distributed solutions to information management problems are known to offer greater flexibility, performance and robustness.

We can distinguish three general categories of systems utilising some sort of middle-agents:

- Centralised, where a single middle-agent is employed.
- Distributed, where multiple co-operating middle-agents are employed.
- Revolutionary, where middle-agents do not exist and all functions typically carried out by middleware are now handled locally by each agent.

2.6.1 The Centralised Approach

In centralised systems a single middle-agent is employed. Typical examples of this approach are:

- Search engines such as Google when interpreted as middle-agents.
- Multi-agent systems such as SHADE, COINS [78] and WARREN [101] where a single middle-agent is used.
- Information dissemination systems like the centralised version of SIFT [145] or XFilter [6].
- Any centralised information integration system e.g., InfoSleuth [12].

2.6.2 The Distributed Approach

In the distributed approach tasks traditionally assigned to a single middle-agent are now handled by multiple co-operating middle-agents. This approach is mentioned in various papers but it is not clear to us if any implemented system of this kind exists (we only know of IDIoMS [123]). What is also surprising is that there is only one published work dealing with theoretical aspects of this problem [65]. It is not clear whether the ideas sketched in [65] have been implemented in a real system.

The IDIoMS System

IDIoMS (Intelligent Distributed Information Management System) is an agent-based system for discovering, managing and presenting information found in a widely distributed network such as the Internet. IDIoMS has been developed jointly by Fujitsu Laboratories and BT Laboratories [123]. The middleware component of IDIoMS is called OAM (Open Agent Middleware) and is described in [94, 130]. In OAM information provider agents advertise the capabilities of information providers to middle-agents using KQML. Capabilities of service providers are expressed in propositional logic. For example, in [94, 130] information providers are database agents and an advertisement can be a disjunction of propositions corresponding to relation names (e.g., [Image]). Every middle-agent that receives an advertisement, stores it in its local database, and also forwards the advertisements to neighbouring middle-agents. [130] explains that a neighbourhood is defined by a graph, which gives the connections of agents (connections are set-up by an administrator of the whole system). When an information request arrives at a middle-agent, it is immediately matched with stored advertisements. Requests are also expressed as formulas in propositional logic e.g., [Image]where BayArea and Motors are relation names [130]. If there is a successful match, the middle-agent sends the request to the correct service agent; service agents, in turn, return appropriate information directly to the requester. If there is not a successful match, the middle-agent forwards the request to another middleware agent. Successful match here means that the propositional logic formula corresponding to a request does not contradict the formula representing an advertisement [130] (this is a rather relaxed form of matching). The same procedure is recursively repeated until an appropriate service provider is found and its address sent back to the requester. The requester then contacts the provider (or providers) directly.

OAM (and IDIoMS) is an interesting system. Its distributed approach allows its authors to claim that it is scalable, robust and extensible. However, the OAM papers [94, 130, 123] leave several interesting questions open. First of all, it is not explained clearly why matching based on consistency is the best choice under all circumstances (to understand this one has to know the exact semantics of advertisements but the semantics are not given clearly in [130]). Also, it seems that there are many ways to optimise the routing of requests and advertisements in OAM (see e.g., the techniques of SIENA [21]) and none of that is studied in [94, 130, 123].

2.6.3 Getting Rid of Middle-agents

In the revolutionary approach middle-agents disappear altogether and their functionality is transferred to each individual agent. This approach has not been implemented so far in any existing multi-agent system. Nice examples of this ap-

proach can be found in peer-to-peer content sharing systems such as Gnutella[30] and Freenet [32].

Freenet

Freenet is a peer-to-peer network of individual nodes connected to each other for the purpose of sharing information in the form of data files [32]. An explicit design goal of Freenet was to avoid the need for centralised middleware. Every Freenet node keeps a datastore containing a number of shared data files, and a dynamic routing table that contains addresses of other nodes and a description of the kind of information they are thought to have. This information is in the form of a key obtained by hashing a textual description of a data file (e.g., agents/brokering/survey.doc).

To retrieve data, a user of Freenet chooses a short descriptive string, which is thought to describe the data file, and hashes it to obtain a file key. The user then sends a request for a file with this key to her own node. Whenever a node receives a request, checks its local datastore first. If the file is found, it is returned together with a note saying that this node was the source of the data. If the file is not found, the routing table is consulted, the neighbouring node with the closest matching key is chosen, and the request is forwarded to it. When the data file is finally found, it is returned to the requester via the same path. Additionally, intermediate nodes save a local copy of the data file together with an entry in the routing table associating the requested key with the data source. Each data request in Freenet is given a hops-to-live count, which is decremented at each node the request goes through successfully in order to reduce message traffic. To prevent requests from going into an infinite loop, Freenet assigns a unique identifier to each request so that a node will never forward a request that goes through it for a second time. To store files, users go through a similar procedure so that nodes will come to be known to other nodes only if they are contributing files to the system [32].

Freenet is an interesting peer-to-peer file sharing system with several positive features claimed by its implementers:

- It is very robust because data is duplicated in several places.
- Its storage strategies lead to clustering of files with similar keys on the same nodes. Also, data files tend to be stored close to their requesters (where close does not denote geographical proximity but rather key closeness).
- The quality of routing requests improves over time because nodes become specialists in locating sets of similar keys.

[30]Gnutella web site: http://www.gnutella.wego.com

- The privacy of information requesters and information providers is protected since a node can never tell whether a requester is actually interested in a data file or is simply forwarding somebody elses request (questions of privacy are further discussed in Section 2.7.

Gnutella

Like Freenet, Gnutella is also a peer-to-peer file sharing system used widely today for sharing music files (e.g., MP3s). Gnutella is very similar to Freenet but it has serious deficiencies [62]. First of all, a Gnutella node broadcasts a request to all of its neighbours thus generating a great amount of network traffic (e.g., in contrast Freenet chooses one of its neighbours to forward a request). Also, the privacy of information requesters and providers is not really protected (e.g., Gnutella messages contain IP addresses, and URLs are returned to information requesters so that they can retrieve the files they desire).

Other Theoretical and Experimental Work

Shehory [115] presents a theoretical analysis of the problem of finding the location of an agent with certain capabilities in open multi-agent systems that are like the peer-to-peer networks of Gnutella and Freenet (i.e., there are no intermediaries so each agent keeps a local list of other agents and their capabilities). [115] considers situations where agents are connected in a lattice-like rectangular graph structure, and shows that dramatic improvements in the number of communication operations is possible by randomly connecting a small number of pairs of nodes in the original graph. Additionally, Shehory gives some evidence that these results apply to larger classes of graphs as well. The practical relevance of this work is that it shows that if there is enough structure in the topology of open multi-agent systems then approaches with no intermediaries may result in very small communication costs. Experimental results that would demonstrate this would be very welcome.

[100] present an experimental study concerned with matchmaking between service providers and consumers without involving middle-agents. The multi-agent system of [100] involves a collection of simple agents that need or can offer services, a set of tasks that each agent needs to complete using outside help, a set of task categories and an algorithm for task matching. There is no centralized control and agents are allowed to communicate only with their neighbours. The notion of neighbourhood is defined by considering each task an agent has to perform as an interface to the agent. At each moment of time, an agent can communicate with only as many agents as it has open tasks that need to be carried out by others. This can be understood by visualizing agents as being distributed spatially so that interfaces of various agents are adjacent. Each agent looks out for needed service providers by carrying out some form

of local search among agents with adjacent interfaces. If another agent with a matching interface is found, then these two agents form a cluster. Clusters are subsequently allowed to rotate so that interfaces that have not been matched can also find matchings. When new matchings are found, clusters are enlarged to include the new service provider agents. This local search procedure seems to perform very well as demonstrated by the simulations of [100]. For example, in a system of 3 tasks per agent, 80 categories and 2000 agents, more than 90 iterations of the search procedure. Unfortunately, the behaviour of the system deteriorates as the number of categories increase beyond 90. [100] also show that the success of the system in finding matchings scales linearly with respect to the number of agents considered.

Although the study of [100] presents interesting arguments in favor of the revolutionary approach to agent middleware, it is not clear whether their experimental settings (e.g., their notion of neighbourhood) can arise in practice. More work is needed in this direction as pointed out in [100].

We conclude this section by conjecturing that research efforts in this area will multiply in the near future as systems like Gnutella attract the attention of academics (see for example, the recent proposal of [62] to use the distributed event notification system Siena [21] to implement a Gnutella-like system). There is also a Peer-to-Peer Working Group[31] devoted to the development of standards for peer-to-peer computing (Intel is a member company, and HP and IBM are supporting companies).

2.7 Middle-agents and Privacy

Protecting the privacy of information producers and consumers has become an important issue in the age of the web. Today, more than ever before, it is very easy for someone to learn a lot about somebody elses preferences or capabilities by carefully observing their behaviour during electronic transactions. However, parties involved in a transaction might not wish to reveal some information about themselves to other participating parties (e.g., a customer browsing the web pages of various electronic shops might not want to reveal any information about her until finally choosing a shop and a product to buy). These observations have resulted in a flurry of interesting research dealing with privacy issues on the web [110].

[39] have categorised various configurations of agents from a privacy viewpoint. This has been achieved by examining the possible flows of information from requesters to providers (and vice versa) via the middle-agents. Preference information can be kept private at the requester, be revealed to some middle-agent, or be known by the provider itself. The same three possibilities exist for

[31] http://www.peer-to-peerwg.org

	Capabilities known by		
Preferences known by	provider only	provider and middle agent	provider and middle and requester
requester only	direct communication (to middle agent)		recommending protocol
requester and middle agent		brokering and recruiting protocol	
requester and middle and provider	blackboard protocol		

Table 2.2: Middle-agent roles from a privacy viewpoint

capability information. From a privacy viewpoint, this leads to nine general protocols for middle-agents (see Table 2.2 below taken from [39]). In the following we analyse three of these cases but similar observations follow for the other six cases. The diligent reader should note that we are using slightly different terminology from [39] (we are using the term recommending instead of matchmaking as in [39]).

In the recommending protocol presented in Section 2.4, capabilities are advertised with a middle-agent (e.g., using the KQML performative advertise). When a request of the form "Give me the names of all agents with capability X" arrives at the middle-agent (using a performative recommend-all), the matchmaker checks the advertisement database and chooses all advertisements that serve the request. Then, the matchmaker provides the requester with the names and Internet addresses of the chosen providers that can satisfy the request. Finally, the requester chooses which providers to query in order to get the information. Thus, using the recommending protocol one can protect the privacy of a requester by not allowing its preferences to become known to providers (until one or more of the providers are contacted by the requester). The blackboard protocol is symmetric to the recommending protocol: only the privacy of providers is protected (the exact details of the protocol are omitted).

Finally, the brokering and recruiting protocols allow one to protect the privacy of both requesters and providers. As explained in Section 2.4, a middle-agent using the brokering protocol keeps track of advertisements coming from information providers and requests coming from information consumers. The middle-agent forwards requests to appropriate providers, collects replies and sends the final information to the requesters. Therefore, requesters and providers never get to know each other if their transactions are handled by a middle-agent using a bro-

kering protocol. The recruiting protocol is similar: the middle-agent forwards the request to appropriate providers, but now the chosen providers reply to the requester directly.

Another dimension, which is not included in Table 2.2, concerns the actual occurrence and/or content of a broker-based transaction. Both recruit and broker, allow the broker to know the actual content exchanged between the requester and the brokered or recruited agent. In such a case, the broker is also aware that the provider agent responded and that the requester is aware of (received) that response. By contrast, when the recommend primitive is used the broker not only does not know the actual response to the requester but it does not even know whether the requester ever placed a request or whether the "recommended" agent responded to it. The implications of these differences, include liability of the broker and knowledge of a third party (the broker) that a transaction occurred and a particular fragment of knowledge has been received by the requester, or sent by a provider [32]. For example, in the case of the Napster P2P system, where the Napster server acts as a broker that essentially processes recommendation requests, the Napster server is not aware of whether requesters actually contact providers and eventually download content from the providers.

2.8 Performance Issues: Scalability, Robustness and Adaptivity

The various protocols for middle-agents discussed in Section 2.4 have been deployed in the context of various implemented applications (e.g., SHADE, COINS [78] and WARREN [101] discussed above). Application requirements will usually determine which protocol is appropriate in each case (e.g., recommending or brokering). However, performance comparisons can guide the choice of an implementer in a design space where more than one approach is possible.

There is little published work which compares different agent middleware solutions regarding scalability, robustness and adaptivity [39, 75, 76]. [39] present a comparison of three protocols (blackboard, recommending and brokering) with respect to scaling up efficiently to large numbers of requests. The formal tool of [39] is an analytical queuing network model [86], which is also validated by experimental results carried out in the context of WARREN [101]. As it might be expected, the results of this analysis show that middle-agents running the brokering protocol can achieve better response times by balancing the load on information providers.

[39] also observe that centralized systems using a single middle-agent are vulnerable to failures of the middle-agent. The solution proposed is to have

[32] On a related note, broker and recruit requests can allow the broker to cache responses from prior queries, for future usage, in cases where the exchanged knowledge has a known "life".

requesters cache results of recommend-one or recommend-all operations so that if the middle-agent fails, the requester can use the locally cached information until the agent functions properly again (this is not possible for brokers). A distributed configuration of middle-agents where robustness would obviously be enhanced has not been considered in [39].

A more interesting scheme for recovery from middle-agent failure has been proposed in [75, 76] and has been implemented in the Adaptive Agent Architecture system [33]. The authors consider multi-agent systems with multiple end-agents and middle-agents. In their proposal whenever an end-agent A advertises its capabilities to a certain middle-agent M, this information is propagated to the rest of the middle-agents which undertake a commitment to connect to end-agent A whenever M disconnects from the rest of the middle-agents. This is essentially a nice way to use teamwork to have a more robust and reliable system of middle-agents. [75] presents the formal underpinnings of this work using the formal concepts of commitment and joint intention, and [76] demonstrate the utility of the recovery scheme experimentally. The findings of [76] are the following:

- The proposed recovery scheme allows the system to function despite middle-agent failure as long as there is at least one middle-agent left in the system.
- There is no significant communication overhead due to the extra messages middle-agents will exchange.
- Adding new middle-agents does not significantly affect the performance of the system.
- The scheme can be implemented so that a specified number of middle-agents is always present in the system.

[39] have also considered the problem of adaptivity of middle-agents in the presence of dynamic changes in the availability of information providers. When providers can come and go, middle-agents using the brokering protocol can achieve better response times by performing careful load balancing. The same effect can be achieved in middle-agents using the recommending protocol but now the responsibility for load balancing relies entirely on information requesters. The related topic of adapting to dynamic changes in the preferences of information requesters or the capabilities of information providers is not handled in depth in [39]. Moukas [96, 97] shows how relevance feedback and techniques from genetic algorithms can be used to improve the adaptivity of similar systems when dynamic changes in preferences of information requesters take place. Relevance feedback techniques for the same problem are also analysed in the elegant framework of [23]. Unfortunately, the techniques of [23, 96, 97] are applicable only to

[33] http://chef.cse.ogi.edu/AAA

keyword-based information retrieval over WWW sources. Thus it is not clear how such algorithms can be employed in the context of [39] where more sophisticated languages for expressing preferences are envisioned (e.g., LARKS [127]).

[38, 117] have also considered execution adaptation for agents that function as information providers (information agents in the terminology of the multi-agent system RETSINA [38]). The main proposal here is that if agents become overloaded with information requests, they can create a clone of themselves and direct some of their load to this new agent. The new agent can execute in a separate agent server using an appropriate implementation language with code migration facilities (e.g., Java). The introspective facilities needed so that an agent can actually notice that it has become overloaded are easily provided in RETSINA. [38, 117] do not mention the related issue of cloning a middle-agent when the available information exceeds the storage capacity available to the agent.

2.9 Conclusions

In this chapter we surveyed recent research on agent middleware with an emphasis on information management applications. We started with the origins of agent middleware (the concepts of mediator and facilitator) and continued with related work up to today. We also discussed some popular Internet systems (search engines, publish/subscribe systems, peer-to-peer systems) and show how they can be casted as multi-agent systems with a central role for agent middleware.

In the rest of this report we concentrate on textual information dissemination as it arises int he architecture sketched in Chapter 1. Our work can be seen as a contribution to the general area of agent middleware surveyed here.

Chapter 3

Data Models and Query Languages for Textual Information Dissemination

In this chapter we concentrate on defining the syntax and semantics of three progressively more extensive data models and query languages for textual information dissemination.

We define formally the models $\mathcal{WP}$, $\mathcal{AWP}$ and $\mathcal{AWPS}$, and their corresponding languages for textual information dissemination in distributed agent systems such as the ones surveyed in Chapter 2. Data model $\mathcal{WP}$ is based on free text and its query language is based on the *boolean model with proximity operators*. The concepts of $\mathcal{WP}$ extend the traditional concept of proximity in IR [10, 27, 28] in a significant way and utilize it in a content language targeted at information dissemination applications. Data model $\mathcal{AWP}$ is based on *attributes* or *fields* with finite-length strings as values. Its query language is an extension of the query language of data model $\mathcal{WP}$. Our work on $\mathcal{AWP}$ complements recent proposals for querying textual information in distributed event-based systems [20, 19] by using linguistically motivated concepts such as *word* and not arbitrary strings. This makes $\mathcal{AWP}$ potentially very useful in some applications (e.g., alert systems for digital libraries or other commercial systems where similar models are supported already for retrieval). Finally, the model $\mathcal{AWPS}$ extends $\mathcal{AWP}$ by introducing a "similarity" operator in the style of modern IR, based on the vector space model [10]. The novelty of our work in this area is the move to query languages much more expressive than the one used in the information dissemination system SIFT [145] where documents and queries are represented by free text. The similarity concept of $\mathcal{AWPS}$ is an extension of the similarity concept pioneered by the system WHIRL [34] and recently also used in the XML query language ELIXIR [30]. We note that both WHIRL and ELIXIR target information retrieval and integration applications, and pay no attention to information dissemination and the concepts/functionality needed in such applications. The work in this chapter

has been previously presented in [41, 69, 73, 72, 70].

3.1 Text Values and Word Patterns

In this section we present our first data model and query language for textual information dissemination. The data model is based on free text which is captured formally by the concept of text value. Our query language is based on the Boolean model with proximity operators. Queries in this model are formalised using the concept of word pattern [28]. The two basic concepts of this section (text values and word patterns) are subsequently used in Section 3.4 to define the attribute-based data model and query language.

We assume the existence of a finite *alphabet* $\mathbf{\Sigma}$. A *word* is a finite non-empty sequence of letters from $\mathbf{\Sigma}$. We also assume the existence of an infinite set of words called the *vocabulary* and denoted by $\mathcal{V}$.

Definition 1 *A* text value *s of length n over vocabulary $\mathcal{V}$ is a total function $s : \{1, 2, \ldots, n\} \rightarrow \mathcal{V}$.*

In other words, a text value s is a finite sequence of words from the assumed vocabulary and $s(i)$ gives the i-th element of s. Text values can be used to represent finite-length strings consisting of words separated by blanks. The length of a text value s (i.e., its number of words) will be denoted by $|s|$.

Example 1 *In all the examples of this chapter, our vocabulary will be the vocabulary of the English language and will be denoted by $\mathcal{E}$. The string*

my holiday in Milos

can be represented by a text value s of length 4 over vocabulary $\mathcal{E}$ with $s(1) =$ my, $s(2) =$ holiday etc. The text value "in Milos" is included in s.

We now give the definition of word-pattern. The definition is given recursively in three stages.

Definition 2 *Let $\mathcal{V}$ be a vocabulary. A* proximity-free word pattern *over vocabulary $\mathcal{V}$ is an expression in any of the following forms:*

1. *w where w is a word in the vocabulary $\mathcal{V}$.*
2. *$\neg wp$ where wp is a proximity-free word pattern.*
3. *$wp_1 \wedge wp_2$ where wp_1, wp_2 are proximity-free word patterns.*
4. *$wp_1 \vee wp_2$ where wp_1, wp_2 are proximity-free word patterns.*
5. *(wp) where wp is a proximity-free word pattern.*

A proximity-free word pattern will be called positive *if it does not contain the negation operator.*

Example 2 *The following are proximity-free word patterns that might appear in queries of a user of a news dissemination system interested in articles on holidays:*

$$holiday, \quad holiday \wedge hotel \wedge beach,$$

$$holiday \wedge Athens \wedge hotel \wedge \neg Hilton, \quad holiday \wedge (beach \vee mountains)$$

Word patterns made of words and the Boolean operators $\wedge, \vee$ and $\neg$ should be understood as in traditional IR systems and modern search engines. These systems typically have a version of negation in the form of binary operator *AND-NOT* which is essentially set difference thus safe (in the database sense of the term [3]). For example, a search engine query wp_1 *AND-NOT* wp_2 will return the set of documents that satisfy wp_1 *minus* these that satisfy wp_2. In our information dissemination setting, there is no problem considering an "unsafe" version of negation since word patterns are checked for satisfaction against a single incoming document. Note that the previous work of [28] has *not* considered negation in its word pattern language (but has considered negation in the query language which supports attributes; see Section 3.4).

We now introduce a new class of word patterns that allows us to capture the concepts of *order* and *distance* between words in a text document. We will assume the existence of a set of *(distance) intervals* $\mathcal{I}$ defined as follows:

$$\mathcal{I} = \{[l, u] : \ l, u \in \mathbb{N}, l \geq 0 \text{ and } l \leq u\} \cup \{[l, \infty) : \ l \in \mathbb{N} \text{ and } l \geq 0\}$$

The symbols $\in$ and $\subseteq$ will be used to denote membership and inclusion in an interval as usual.

The following definition uses intervals to impose lower and upper bounds on distances between word patterns.

Definition 3 *Let $\mathcal{V}$ be a vocabulary. A* proximity word pattern *over vocabulary $\mathcal{V}$ is an expression*

$$wp_1 \prec_{i_1} wp_2 \prec_{i_2} \cdots \prec_{i_{n-1}} wp_n$$

where $wp_1, wp_2, \ldots, wp_n$ are positive proximity-free word patterns over $\mathcal{V}$ and $i_1, i_2, \ldots, i_{n-1}$ are intervals from the set $\mathcal{I}$. The symbols $\prec_i$ where $i \in \mathcal{I}$ are called proximity operators. *The number of proximity-free word patterns in a proximity word pattern (i.e., n above) is called its* size.

Example 3 *The following are proximity word patterns:*

$$Holiday \prec_{[0,0]} Inn, \quad The \prec_{[0,0]} Mini \prec_{[0,0]} Palace \prec_{[0,0]} Hotel,$$

$$luxurious \prec_{[0,3]} hotel, \quad luxurious \prec_{[0,3]} (hotel \vee apartment),$$

$$holiday \prec_{[0,10]} beach \prec_{[0,10]} (clean \wedge sandy), \quad hotel \prec_{[0,\infty)} view$$

The proximity word pattern $wp_1 \prec_{[l,u]} wp_2$ stands for "word pattern wp_1 is *before* wp_2 and is separated by wp_2 by *at least* l and *at most* u *words*". In the above example $luxurious \prec_{[0,3]} hotel$ denotes that the word "hotel" appears before word "luxurious" and at a distance of at least 0 and at most 3 words. The word pattern $Holiday \prec_{[0,0]} Inn$ denotes that the word "Holiday" appears exactly before word "Inn" so this is a way to encode the string "Holiday Inn". We can also have arbitrarily long sequences of proximity operators with similar meaning (see the examples above). Note that proximity-free subformulas in proximity word-patterns can be more complex than just simple words (but negation is *not* allowed; this restriction will be explained below). This makes proximity-word patterns a very expressive notation.

Traditional IR systems have proximity operators kW and kN where k is a natural number. The proximity word pattern $wp_1\ kW\ wp_2$ stands for "word pattern wp_1 is before wp_2 and is separated by wp_2 by at most k words". In our work this can be captured by $wp_1 \prec_{[0,k]} wp_2$. The operator kN is used to denote distance of at most k words where the order of the involved patterns does not matter. In our framework the expression $wp_1\ kN\ wp_2$ can be approximated by $wp_1 \prec_{[0,k]} wp_2 \vee wp_2 \prec_{[0,k]} wp_1$ as in [28].[1] In addition to the above operators, our framework allows the expression of simple order constraints between words using operators $\prec_{[0,\infty]}$. This feature does not appear to be useful immediately, but it is required if we want to be able to express some word patterns that are logically entailed (see Section 3.2). Order constraints of the form $\prec_{[0,\infty]}$ between various text structures are also present in more advanced text model proposals such as the model of proximal nodes of [98].

Note that proximity operators are very useful and have been popular with sophisticated users of search engines. However, only limited forms of them are in use in current popular search engines (e.g., Altavista's operator $NEAR$ means word-distance 10, Lycos' operator $NEAR$ means word-distance 25, Infoseek used to have a more sophisticated facility but not any more[2]). Proximity operators have also been implemented in other systems such as freeWAIS[3] [104] and INQUERY [18]. There are also advanced IR models such as the model of proximal nodes [98] with proximity operators between arbitrary structural components of a document (e.g., paragraphs or sections). Query languages for XML will probably be the next place to see proximity operators (the recent work [146] contains such an example).

Definition 4 *Let $\mathcal{V}$ be a vocabulary. A* word pattern *over vocabulary $\mathcal{V}$ is an expression in any of the following categories:*

[1] [28] (page 23) gives an example that demonstrates why these two expressions are not equivalent given the meaning of operator kN. The example involves a text value and word patterns with overlapping positions in that text value hence the difference.

[2] www.searchlores.org

[3] http://ls6-www.informatik.uni-dortmund.de/ir/projects/freeWAIS-sf/

1. *a proximity-free word pattern over* $\mathcal{V}$
2. *a proximity word pattern over* $\mathcal{V}$
3. $wp_1 \wedge wp_2$ *where* wp_1, wp_2 *are word patterns.*
4. $wp_1 \vee wp_2$ *where* wp_1, wp_2 *are word patterns.*
5. (wp) *where* wp *is a word pattern.*

A word pattern will be called positive *if its proximity-free subformulas are positive.*

Example 4 *The following are word patterns of the most general kind we allow:*

$$holiday \wedge (hotel \prec_{[0,10]} (cheap \wedge clean)),$$

$$holiday \wedge (luxurious \prec_{[0,0]} hotel) \wedge \neg Hilton,$$

$$holiday \wedge (luxurious \prec_{[0,0]} hotel) \wedge (Holiday \prec_{[0,0]} Inn),$$

$$Viterbo \wedge ((Dolce \prec_{[0,0]} Vita \prec_{[0,0]} Hotel) \vee (The \prec_{[0,0]} Mini \prec_{[0,0]} Palace \prec_{[0,0]} Hotel)),$$

$$holiday \wedge (luxurious \prec_{[0,0]} hotel \prec_{[0,5]} beach)$$

We have here completed the definition of the concept of word pattern. We now turn to defining their semantics.

3.2 Semantics

We now give meaning to the expressions that define word patterns. First, we define what it means for a text value to satisfy a proximity-free word pattern.

Definition 5 *Let* $\mathcal{V}$ *be a vocabulary,* s *a text value over* $\mathcal{V}$ *and* wp *a proximity-free word pattern over* $\mathcal{V}$*. The concept of* s satisfying wp *(denoted by* $s \models_P wp$*) is defined as follows:*

1. *If* wp *is a word of* $\mathcal{V}$ *then* $s \models wp$ *iff there exists* $p \in \{1, \ldots, |s|\}$ *and* $s(p) = wp$*.*
2. *If* wp *is of the form* $\neg wp_1$ *then* $s \models wp$ *iff* $s \not\models wp_1$*.*
3. *If* wp *is of the form* $wp_1 \wedge wp_2$ *then* $s \models wp$ *iff* $s \models wp_1$ *and* $s \models wp_2$*.*
4. *If* wp *is of the form* $wp_1 \vee wp_2$ *then* $s \models wp$ *iff* $s \models wp_1$ *or* $s \models wp_2$*.*
5. *If* wp *is of the form* (wp_1) *then* $s \models wp$ *iff* $s \models wp_1$*.*

The above definition mirrors the definition of satisfaction for Boolean logic [102]. This will allow us to draw on a lot of related results in the rest of this chapter.

Example 5 *Let s be the following text value:*

During our holiday in Milos we stayed in a luxurious hotel by the beach

Then $s \models holiday \wedge Milos$.

The following definition captures the notion of a set of positions in a text value containing exactly the words that contribute to the satisfaction of a proximity-free word pattern. This notion is then used to define satisfaction of proximity word patterns.

Definition 6 *Let $\mathcal{V}$ be a vocabulary, s a text value over $\mathcal{V}$, wp a proximity-free word pattern over $\mathcal{V}$, and P a subset of $\{1, \ldots, |s|\}$. The concept of s* satisfying *wp* with set of positions *P (denoted by $s \models_P wp$) is defined as follows:*

1. *If wp is a word of $\mathcal{V}$ then $s \models_P wp$ iff there exists $x \in \{1, \ldots, |s|\}$ such that $P = \{x\}$ and $s(x) = wp$.*
2. *If wp is of the form $wp_1 \wedge wp_2$ then $s \models_P wp$ iff there exist sets of positions $P_1, P_2 \subseteq \{1, \ldots, |s|\}$ such that $s \models_{P_1} wp_1$, $s \models_{P_2} wp_2$ and $P = P_1 \cup P_2$.*
3. *If wp is of the form $wp_1 \vee wp_2$ then $s \models_P wp$ iff $s \models_P wp_1$ or $s \models_P wp_2$.*
4. *If wp is of the form (wp_1) then $s \models_P wp$ iff $s \models_P wp_1$.*

Now we define what it means for a text value to satisfy a proximity word pattern.

Definition 7 *Let $\mathcal{V}$ be a vocabulary, s a text value over $\mathcal{V}$ and wp a proximity word pattern over $\mathcal{V}$ of the form*

$$wp_1 \prec_{i_1} wp_2 \prec_{i_2} \cdots \prec_{i_{n-1}} wp_n.$$

Then $s \models wp$ iff there exist sets $P_1, P_2, \ldots, P_n \subseteq \{1, \ldots, |s|\}$ such that $s \models_{P_j} wp_j$ and $min(P_j) - max(P_{j-1}) - 1 \in i_{j-1}$ for all $j = 2, \ldots, n$ (the operators max and min have the obvious meaning).

Example 6 *The text value*

During our holiday in Milos we stayed in a luxurious hotel by the beach

satisfies the following word patterns:

$luxurious \prec_{[0,0]} hotel \prec_{[0,5]} beach$, $luxurious \prec_{[0,0]} (hotel \vee apartment) \prec_{[0,5]} beach$,

$$(holiday \wedge Milos) \prec_{[0,10]} luxurious \prec_{[0,0]} hotel$$

The sets of positions required by the definition are for the first and second word pattern $\{10\}$, $\{11\}$ and $\{14\}$, and for the third one $\{3, 5\}$, $\{10\}$ and $\{11\}$.

If the structure of *wp* falls under the four cases of our most general definition (Definition 4), satisfaction is similarly defined in a recursive way as in Definition 5 (for Cases 1, 3 and 4) and Definition 7 (for Case 2).

Example 7 *The text value*

During our holiday in Milos we stayed in a luxurious hotel by the beach

satisfies word pattern $holiday \wedge (luxurious \prec_{[0,0]} hotel \prec_{[0,5]} beach)$.

We have here completed the definition of the concept of satisfaction of a word pattern by a text value. Let us now compare our development with the approach of Chang and colleagues presented in papers [27, 28, 26]. To the best of our knowledge, these papers contain the only comprehensive treatment of proximity word patterns that exists in the literature. The development of proximity word patterns in [27, 28, 26] follows closely the IR tradition i.e., operators kW and kN (already mentioned above) are used together with the boolean operators AND and OR. These operators can be intermixed in arbitrary ways (e.g., $((w_1\ AND\ (w_2\ (8W)\ w_3))\ (10W)\ w_4)$ where w_1, w_2, w_3, w_4 are words is a legal expression), and the result of their evaluation on document databases is defined in an algebraic way (page 9 of [28]). We have opted for an approach which is more in the spirit of Boolean logic, allows negation and carefully distinguishes word patterns with and without proximity operators. Thus operators cannot be mixed in arbitrary ways but only as prescribed by Definition 4. This leads to a simpler language because cumbersome (and not especially useful) constructions such as the above are avoided. In the spirit of Boolean logic, a word pattern allows us to distinguish between text values: these that satisfy it, and these that do not. The presence of a word w in some word pattern means "word w is in the text value that satisfies it". Boolean operators are then given the usual semantics as in Boolean logic. Proximity operators are given semantics separately as we have done above. Negation is not allowed to appear inside proximity formulas because it does not seem to offer us any useful capabilities in terms of expressing interesting queries.

We now continue our development with the concept of satisfiable word patterns.

Definition 8 *A word pattern wp is called* satisfiable *if there is a text value s that satisfies it. Otherwise it is called* unsatisfiable.

Example 8 *The following word patterns are satisfiable:*

$$holiday \wedge hotel,\ holiday \vee vacation,\ Holiday \prec_{[0,0]} Inn$$

The following word patterns are unsatisfiable:

$$holiday \wedge \neg holiday,\ (nice \prec_{[0,0]} holiday) \wedge \neg holiday$$

Definition 9 *Let wp_1 and wp_2 be word patterns. We will say that wp_1* entails *wp_2 (denoted by $wp_1 \models wp_2$) iff for every text value s such that $s \models wp_1$, we have $s \models wp_2$. If $wp_1 \models wp_2$, we also say that wp_2* subsumes *wp_1. If $wp_1 \models wp_2$ and $wp_2 \models wp_1$ then wp_1 and wp_2 are called* equivalent *(denoted by $wp_1 \equiv wp_2$).*

Example 9 *The word pattern $holiday \wedge hotel$ entails $holiday$. The word pattern $holiday \wedge (luxurious \prec_{[0,0]} hotel \prec_{[0,5]} beach)$ entails the word pattern $holiday \wedge (hotel \prec_{[0,10]} beach) \wedge luxurious$. The word pattern $vacation$ subsumes the word pattern $sun \wedge vacation \wedge beach$. Similarly, $vacation \vee cruise$ subsumes $cruise$. The word pattern $vacation \wedge (beach \vee mountains)$ is equivalent to $(vacation \wedge beach) \vee (vacation \wedge mountains)$.*

The notions of entailment, subsumption and equivalence defined above naturally correspond to standard notions of Boolean logic. If we consider word patterns to be queries, we also have a natural correspondence with the same notions of relational database theory [25, 67]. In the case of proximity-free word patterns the correspondence with Boolean logic is exact, thus the only new feature in our enquiry is the presence of proximity word patterns. Subsumption and minimal subsuming queries involving proximity word patterns have also been studied in [28]. Our work differs from [28] in the following ways: (i) we deal with proximity in a slightly different way (our proximity operators are more general and we do not mix proximity and boolean operators in arbitrary ways), and (ii) the problem in [28] is one of integration thus capabilities of data sources have to be taken into account in relevant definitions. However, many of the results of [28] are important to us and appropriate credits will be given in the rest of the chapter.

Let us close this section by discussing why negation is not allowed to apply to proximity word patterns (Definition 4). If negation is allowed, we should not expect that the resulting formula could always be rewritten equivalently into one where negation is moved somehow inwards as in the case of Boolean logic. The interested reader can be persuaded by trying to do this for the following formula (which illegal in our model):

$$\neg(luxurious \prec_{[0,3]} hotel \prec_{[0,3]} beach)$$

If we restrict our attention to proximity formulas with a single proximity operator, this restriction can easily be lifted. The word pattern $\neg(luxurious \prec_{[0,3]} hotel)$ is equivalent to

$$\neg luxurious \vee \neg hotel \vee (hotel \prec_{[0,\infty]} luxurious) \vee (luxurious \prec_{[4,\infty]} hotel).$$

3.3 Some Properties of Satisfaction and Entailment

In this section we present some results that are easy to show and give some important facts about the semantic notions introduced in Section 3.2. These

results are fundamental for any system of information dissemination based on the data model we develop here.

Proposition 1 *Let wp_1 and wp_2 be word patterns and wp_2 is proximity-free. $wp_1 \models wp_2$ iff $wp_1 \wedge \neg wp_2$ is unsatisfiable.*

The above proposition gives the usual relation between entailment and unsatisfiability of Boolean logic as it should be stated in our framework. Note that wp_2 is required to be proximity-free so that the negation operator can be applied.

The next proposition allows us to compute proximity-free word patterns entailed by a given proximity word pattern. A similar result is stated in [28] for the case of IR operator kW.

Proposition 2 *Let $wp_0, wp_1, wp_2, \ldots, wp_n$ be proximity-free word patterns. If $\bigvee_{k=1}^{n} wp_k \models wp_0$ then*

$$wp_1 \prec_{i_1} wp_2 \prec_{i_2} \cdots \prec_{i_{n-1}} wp_n \models wp_0$$

Example 10 *From the above proposition we have that luxurious $\prec_{[0,0]}$ hotel entails hotel. Also, hotel $\prec_{[0,20]}$ (view $\wedge$ sunset) entails view $\vee$ cliff.*

The next three propositions allows us to compute proximity word patterns entailed by a given proximity word pattern. The first one enables us to compute an entailed word pattern by eliminating the first or the last proximity-free subformula in a given proximity word pattern.

Proposition 3 *Let $wp_1, wp_2, \ldots, wp_{n-1}, wp_n$ be proximity-free word patterns, and wp the proximity word pattern*

$$wp_1 \prec_{i_1} wp_2 \prec_{i_2} \cdots \prec_{i_{n-2}} wp_{n-1} \prec_{i_{n-1}} wp_n.$$

Then wp entails

$$wp_2 \prec_{i_2} \cdots \prec_{i_{n-2}} wp_{n-1} \prec_{i_{n-1}} wp_n$$

and

$$wp_1 \prec_{i_1} wp_2 \prec_{i_2} \cdots \prec_{i_{n-2}} wp_{n-1}.$$

Example 11 *Let wp be*

$$luxurious \prec_{[0,0]} hotel \prec_{[2,5]} (sandy \wedge clean) \prec_{[0,0]} beach.$$

Then wp entails

$$hotel \prec_{[2,5]} (sandy \wedge clean) \prec_{[0,0]} beach$$

and

$$luxurious \prec_{[0,0]} hotel \prec_{[2,5]} (sandy \wedge clean).$$

The next proposition enables us to compute an entailed word pattern by eliminating any proximity-free subformula other than the first or the last in a proximity word pattern. In this case, a new proximity operator must be computed for the resulting formula.

Proposition 4 *Let* $wp_1, wp_2, \ldots, wp_{k-1}, wp_k, wp_{k+1}, \ldots, wp_n$ *be proximity-free word patterns, and wp the proximity word pattern*

$$wp_1 \prec_{i_1} wp_2 \prec_{i_2} \cdots \prec_{i_{k-2}} wp_{k-1} \prec_{i_{k-1}} wp_k \prec_{i_k} wp_{k+1} \prec_{i_{k+1}} \cdots \prec_{i_{n-1}} wp_n$$

where wp_k *is a word or a conjunction of words. If* $n > 1$ *and* $2 \leq k \leq n-1$ *then wp entails*

$$wp_1 \prec_{i_1} wp_2 \prec_{i_2} \cdots \prec_{i_{k-2}} wp_{k-1} \prec_{i'} wp_{k+1} \prec_{i_{k+1}} \cdots \prec_{i_{n-1}} wp_n$$

where i' *is an interval defined as follows. Let* i_{k-1} *be* $[l_{k-1}, u_{k-1}]$ *and* i_k *be* $[l_k, u_k]$. *If* $wp_k \equiv w$ *where w is a word then* $i' = [l_{k-1} + l_k + 1, u_{k-1} + u_k + 1]$. *Otherwise,* $wp_k \equiv w_1 \wedge \cdots \wedge w_m$ *where* $m > 1$ *and* $w_1, \ldots, w_m$ *are distinct words and* $i' = [l_{k-1} + l_k + m, \infty)$. *Interval* i' *is defined similarly if the right endpoint of* i_{k-1} *or* i_k *is* ∞.

Example 12 *Let wp be*

$$luxurious \prec_{[0,0]} hotel \prec_{[2,5]} (sandy \wedge clean) \prec_{[0,0]} beach.$$

Then wp entails

$$luxurious \prec_{[3,6]} (sandy \wedge clean) \prec_{[0,0]} beach$$

and

$$luxurious \prec_{[0,0]} hotel \prec_{[4,\infty)} beach.$$

The new proximity operator in the above word pattern is $\prec_{[4,\infty)}$ *because we cannot put an upper bound on the number of words between* $s(min(P_3))$ *and* $s(max(P_3))$ *for any text value s such that* $s \models_{P_3} sandy \wedge clean$ *(see Definition 7).*

It is interesting to point out that a result similar to the one stated in Proposition 4, is *not* included in [28] for the case of the IR operator kW although it seems to be natural. For example, the word pattern

$$luxurious \prec_{[0,0]} (small \wedge hotel) \prec_{[0,5]} beach$$

can be expressed by

$$luxurious\ (0W)\ (small \wedge hotel)\ (5W)\ beach$$

in the framework of [28], and it entails word pattern $luxurious \prec_{[2,\infty)} beach$. But this entailed word pattern cannot be expressed in the language of [28], thus the class of proximity word patterns studied there is not closed with respect to entailment.

The next proposition allows us to compute an entailed formula by weakening a proximity constraint.

Proposition 5 *Let $wp_1, wp_2, \ldots, wp_k, wp_{k+1}, \ldots, wp_n$ be proximity-free word patterns, and wp the proximity word pattern*

$$wp_1 \prec_{i_1} wp_2 \prec_{i_2} \cdots \prec_{i_{k-1}} wp_k \prec_{i_k} wp_{k+1} \prec_{i_{k+1}} \cdots \prec_{i_{n-1}} wp_n.$$

If $k \in \mathbb{N}$ and $1 \leq k \leq n-1$ then wp entails

$$wp_1 \prec_{i_1} wp_2 \prec_{i_2} \cdots \prec_{i_{k-1}} wp_k \prec_{i'} wp_{k+1} \prec_{i_{k+1}} \cdots \prec_{i_{n-1}} wp_n$$

for every interval i' such that $i \subseteq i'$.

Example 13 *Let wp be*

$$luxurious \prec_{[0,0]} hotel \prec_{[2,5]} (sandy \wedge clean) \prec_{[0,0]} beach.$$

Then wp entails

$$luxurious \prec_{[0,5]} hotel \prec_{[2,5]} (sandy \wedge clean) \prec_{[0,0]} beach.$$

The usual laws for equivalent Boolean expressions with connectives $\neg, \wedge$ and $\vee$ hold for the case of proximity-free word patterns (commutativity, associativity etc. – see Proposition 4.1 in [102]). The following equivalence which is easy to show states that proximity operators distribute over $\vee$. However, proximity operators do not *distribute* over $\wedge$ i.e., $(luxurious \wedge hotel) \prec_{[2,2]} wp_3$ is not equivalent to

$$(luxurious \prec_{[2,2]} beach) \wedge (hotel \prec_{[2,2]} beach)$$

because there is a text value (e.g., "*a luxurious hotel on the beach*") that satisfies the former but not the latter formula [28].

Proposition 6 *Let $wp_1, \ldots, wp_k, \ldots, wp_n$ be proximity-free word patterns, and wp the proximity word pattern*

$$wp_1 \prec_{i_1} \cdots \prec_{i_{k-1}} wp_k \prec_{i_k} \cdots \prec_{i_{n-1}} wp_n.$$

If word pattern wp_k is of the form $\phi \vee \psi$ then wp is equivalent to

$$(wp_1 \prec_{i_1} \cdots \prec_{i_{k-1}} \phi \prec_{i_k} \cdots \prec_{i_{n-1}} wp_n)$$

$$\vee$$

$$(wp_1 \prec_{i_1} \cdots \prec_{i_{k-1}} \psi \prec_{i_k} \cdots \prec_{i_{n-1}} wp_n).$$

For the case of IR operators kW and kN, the above law was recently presented in [28]. The curious reader can also see the rather archaic [93] (credited with the proximity laws in [28]) for related discussion that took place a long time ago.

Example 14 *The word pattern*

$$luxurious \prec_{[0,0]} (hotel \vee bungalow) \prec_{[2,10]} beach$$

is equivalent to

$$(luxurious \prec_{[0,0]} hotel \prec_{[2,10]} beach) \vee (luxurious \prec_{[0,0]} bungalow \prec_{[2,10]} beach).$$

Proposition 7 *Let $wp_1, \ldots, wp_k, \ldots, wp_n$ be proximity-free word patterns, and wp the proximity word pattern*

$$wp_1 \prec_{i_1} \cdots \prec_{i_{k-1}} wp_k \prec_{i_k} \cdots \prec_{i_{n-1}} wp_n.$$

If word pattern wp_k is of the form $w_1 \wedge \cdots \wedge w_m$ where $w_1, \ldots, w_m$ are words then wp is equivalent to

$$\bigvee_{(l_1,\ldots,l_m) \text{ is a permutation of } (w_1,\ldots,w_m)} l_1 \prec_{[0,\infty)} \cdots \prec_{[0,\infty)} l_m.$$

Example 15 *The word pattern*

$$(clean \wedge cheap) \prec_{[0,5]} hotel \prec_{[2,5]} beach$$

is equivalent to

$$clean \prec_{[0,\infty)} cheap \prec_{[0,5]} hotel \prec_{[2,5]} beach$$
$$\vee$$
$$cheap \prec_{[0,\infty)} clean \prec_{[0,5]} hotel \prec_{[2,5]} beach.$$

3.3.1 Normal Forms

We now define the concepts of atomic word patterns, conjunctive/disjunctive word patterns, and the conjunctive/disjunctive normal forms.

Definition 10 *A word pattern is called* atomic *if it is a word, a negated word or a proximity word pattern $w_1 \prec_{i_1} \cdots \prec_{i_{n-1}} w_n$ where $w_1, \ldots, w_n$ are words. A word pattern is called* conjunctive *(resp.* disjunctive*) if it is a conjunction (resp. disjunction) of atomic word patterns. A word pattern is in* conjunctive normal form (CNF) *(resp.* disjunctive normal form (DNF)*) if it is a conjunction (resp. disjunction) of disjunctive (resp. conjunctive) word patterns.*

The following easy proposition can be proved by induction on the structure of a word pattern given the laws discussed above.

Proposition 8 *Every word pattern is equivalent to a word pattern in CNF and a word pattern in DNF.*

For the case of IR operators kW and kN, the above proposition has been stated in [28].

3.4 An Attribute-Based Data Model and Query Language

Now that we have studied text values and word patterns in great detail, we are ready to define our second data model and query language. This data model for text documents is based on *attributes* or *fields* with finite-length strings as values. Attributes are used to encode information such as author, title, date, body of text and so on. This simple data model is restrictive since it offers a rather flat view of a text document, but it has wide applicability as we will show below.

We start our formal development by defining the concepts of document schema and document. Throughout the rest of this chapter we assume the existence of a countably infinite set of attributes $\mathbf{U}$ called the *attribute universe*.

Definition 11 *A* document schema $\mathcal{D}$ *is a pair* $(\mathcal{A}, \mathcal{V})$ *where* $\mathcal{A}$ *is a subset of the attribute universe* $\mathbf{U}$ *and* $\mathcal{V}$ *is a vocabulary.*

Example 16 *An example of a document schema for a news dissemination application is* $\mathcal{D} = (\{SENDER,\ EMAIL,\ BODY\}, \mathcal{E})$.

Definition 12 *Let* $\mathcal{D}$ *be a document schema. A* document *d over schema* $(\mathcal{A}, \mathcal{V})$ *is a set of attribute-value pairs* (A, s) *where* $A \in \mathcal{A}$, *s is a text value over* $\mathcal{V}$, *and there is at most one pair* (A, s) *for each attribute* $A \in \mathcal{A}$.

Example 17 *The following is a document over the schema of Example 16:*

$\{$ $(SENDER,$ "$Manolis\ Koubarakis$"$)$, $(EMAIL,$ "$manolis@ced.tuc.gr$"$)$,

$(BODY,$ "$During\ our\ holiday\ in\ Milos\ we\ stayed\ in\ a\ wonderful\ hotel\ by\ the\ beach$"$)$ $\}$

Before we proceed, it is instructive to compare our text document data model with other models available in the literature. Simple text data models like ours have been popular with database researchers for some time due to their obvious connections to the relational data model [10, 95]. The model of this section is essentially the model of [27, 28, 26] with a different class of proximity word patterns as explained in Section 3.2. The database community has now moved to semi-structured XML-based models for the representation of textual information [2] and our work can naturally be extended to follow this important research avenue.

The model of this section appears to be rather simple in comparison with traditional relational or object-oriented models as they could have been used for storing textual data [95]. However our model is not as rigid as the relational or object-oriented data model so it enables a useful form of heterogeneity in the limited structure allowed. In this respect, our model is similar to recently proposed network directory models [64] although it does not contain a notion

of class or hierarchy and all attributes have the same type. In fact, one could imagine extending network directory data models (such as the one in [64]) to allow for a text data type, and also extend the relevant query languages (such as the language $\mathcal{L}_0$ of [64]) to have the boolean and proximity operators discussed in Section 3.1.

As it has been first discussed in [28], simple attribute-value data models like ours have also been popular with commercial information retrieval services (e.g., Dialog[4] or Lexis-Nexis[5]), digital library protocols (e.g., Z39.50[6]), resource discovery systems [89], and metadata standardisation activities for digital collections (e.g., Dublin Core[7]). The results of this work will be useful to any of these systems if an information integration and dissemination service like the one discussed in the application scenario of Section 2.1 is developed.

Our document model and query language is also related to the data model for news group warehouses recently proposed in [61]. This is rather reasonable given that news dissemination is one of the motivating applications of our work. The query language presented in [61] allows selection of news articles based on conditions involving $=$ and $\sqsupseteq$ operators like we do in our work. Also, the newsgroup maintenance problem as defined in [61] can be rephrased in our framework as the problem of finding which posted profiles are satisfied by a document that has just arrived.

The syntax of our query language is given by the following recursive definition.

Definition 13 *Let $\mathcal{D} = (\mathcal{A}, \mathcal{V})$ be a document schema. A* query *over $\mathcal{D}$ is a formula in any of the following forms:*

1. *$A \sqsupseteq wp$ where $A \in \mathcal{A}$ and wp is a* positive *word pattern over $\mathcal{V}$. The formula $A \sqsupseteq wp$ can be read as "A contains word pattern wp".*
2. *$A = s$ where $A \in \mathcal{A}$ and s is a text value over $\mathcal{V}$.*
3. *$\neg\phi$ where ϕ is a query containing no proximity word patterns.*
4. *$\phi_1 \vee \phi_2$ where ϕ_1 and ϕ_2 are queries.*
5. *$\phi_1 \wedge \phi_2$ where ϕ_1 and ϕ_2 are queries.*

Example 18 *The following are queries over the schema of Example 16:*

$$SENDER \sqsupseteq (John \prec_{[0,2]} Smith),$$

$$(BODY \sqsupseteq (Milos \wedge (hotel \prec_{[0,5]} beach))) \wedge \neg SENDER = \text{“}John\ Smith\text{”}$$

[4] www.dialog.com
[5] www.lexis-nexis.com
[6] http://www.niso.org/standard.html
[7] www.dublincore.org

3.4.1 Semantics

Let us now define the semantics of the above query language in our dissemination setting. We start by defining when a document satisfies a query.

Definition 14 *Let $\mathcal{D}$ be a document schema, d a document over $\mathcal{D}$ and ϕ a query over $\mathcal{D}$. The concept of document d* satisfying *query ϕ (denoted by $d \models \phi$) is defined as follows:*

1. *If ϕ is of the form $A \sqsupseteq wp$ then $d \models \phi$ iff there exists a pair $(A, s) \in d$ and $s \models wp$.*
2. *If ϕ is of the form $A = s$ then $d \models \phi$ iff there exists a pair $(A, s) \in d$.*
3. *If ϕ is of the form $\neg\phi_1$ then $d \models \phi$ iff $d \not\models \phi_1$.*
4. *If ϕ is of the form $\phi_1 \wedge \phi_2$ then $d \models \phi$ iff $d \models \phi_1$ and $d \models \phi_2$.*
5. *If ϕ is of the form $\phi_1 \vee \phi_2$ then $d \models \phi$ iff $d \models \phi_1$ or $d \models \phi_2$.*

Example 19 *The first query of Example 18 is not satisfied by the document of Example 17 while the second one is satisfied.*

We now define the concepts of query satisfiability, entailment and equivalence.

Definition 15 *Let ϕ be a query over schema $\mathcal{D}$. If there is a document d over $\mathcal{D}$ such that $d \models \phi$ then ϕ is called* satisfiable *otherwise it is called* unsatisfiable.

Example 20 *The queries*

$$SENDER = \text{“}Jones\text{”} \wedge SENDER = \text{“}King\text{”},$$

$$SENDER \sqsupseteq Jones \wedge \neg SENDER \sqsupseteq Jones,$$

$$SENDER = \text{“}Michael\ Jones\text{”} \wedge \neg SENDER \sqsupseteq Jones$$

are unsatisfiable.

Definition 16 *Let ϕ_1 and ϕ_2 be queries over schema $\mathcal{D}$. Then ϕ_1* entails *ϕ_2 (denoted by $\phi_1 \models \phi_2$) iff every document d over $\mathcal{D}$ that satisfies ϕ_1 also satisfies ϕ_2. If ϕ_1 entails ϕ_2 then we also say that ϕ_2* subsumes *ϕ_1. We will say that ϕ_1* is equivalent *to ϕ_2 (denoted by $\phi_1 \equiv \phi_2$) iff ϕ_1 entails ϕ_2 and vice versa.*

Example 21 *The query*

$$(BODY \sqsupseteq Milos) \wedge (BODY \sqsupseteq Milos \wedge (hotel \prec_{[0,3]} beach)) \wedge \neg SENDER = \text{“}John\ Smith\text{”}$$

is equivalent to

$$(BODY \sqsupseteq Milos \wedge (hotel \prec_{[0,5]} beach)) \wedge \neg SENDER = \text{“}John\ Smith\text{”}.$$

The latter query entails (equivalently: is subsumed by) $BODY \sqsupseteq Milos$.

We now define the concepts of atomic queries, conjunctive/disjunctive queries, and conjunctive/disjunctive normal forms.

Definition 17 *A query is called* atomic *if it is in one of the following forms:* $A = s, \neg A = s, A \sqsupseteq w, \neg A \sqsupseteq w$ *or* $A \sqsupseteq wp$ *where* s *is a text value,* w *is a word and* wp *is an atomic proximity word pattern. A query is called* conjunctive *(resp.* disjunctive*) if it is a conjunction (resp. disjunction) of atomic queries. A query is in* conjunctive normal form (CNF) *(resp.* disjunctive normal form (DNF)*) if it is a conjunction (resp. disjunction) of disjunctive (resp. conjunctive) queries.*

The following proposition is easy to see.

Proposition 9 *Let* A *be an attribute and* wp_1, wp_2 *be word patterns. Then the following equivalences hold:*

1. $A \sqsupseteq (wp_1 \wedge wp_2) \equiv (A \sqsupseteq wp_1) \wedge (A \sqsupseteq wp_2)$
2. $A \sqsupseteq (wp_1 \vee wp_2) \equiv (A \sqsupseteq wp_1) \vee (A \sqsupseteq wp_2)$
3. $\neg(A \sqsupseteq (wp_1 \wedge wp_2)) \equiv (\neg A \sqsupseteq wp_1) \vee (\neg A \sqsupseteq wp_2)$
4. $\neg(A \sqsupseteq (wp_1 \vee wp_2)) \equiv (\neg A \sqsupseteq wp_1) \wedge (\neg A \sqsupseteq wp_2)$

In Cases 3 and 4, wp_1 *and* wp_2 *are assumed not to contain proximity operators.*

From Proposition 8 and 9, we now have the following result which closes this chapter.

Proposition 10 *Every query is equivalent to a query in DNF and a query in CNF.*

3.5 Extending $\mathcal{AWP}$ with Similarity

Let us now define our third data model $\mathcal{AWPS}$ and its query language. $\mathcal{AWPS}$ extends $\mathcal{AWP}$ with the concept of *similarity* between two text values (the letter $\mathcal{S}$ stands for similarity). The idea here is to have a "soft" alternative to the "hard" operator $\sqsupseteq$. This operator is very useful for queries such as "I am interested in documents sent by John Brown" which can be written in $\mathcal{AWP}$ as

$$SENDER \sqsupseteq (John \prec_{[0,0]} Brown)$$

but it might not be very useful for queries "I am interested in documents about the use of ideas from agent research in the area of information dissemination".

The desired functionality can be achieved by resorting to an important tool of modern IR: the *weight* of a word as defined in the Vector Space Model (VSM) [10, 91, 141]. In VSM, documents (text values in our terminology) are conceptually

represented as vectors. If our vocabulary consists of n distinct words then a text value s is represented as an n-dimensional vector of the form $(\omega_1, \ldots, \omega_n)$ where ω_i is the weight of the i-th word (the weight assigned to a non-existent word is 0). With a good weighting scheme, the VSM representation of a document can be a surprisingly good model of its semantic content in the sense that "similar" documents have very close semantic content. This has been demonstrated by many successful IR systems recently (see for example, WHIRL [34]). [8]

In VSM, the weight of a word is computed using the heuristic of assigning higher *weights* to words that are frequent in a document and *infrequent* in the collection of documents available. This heuristic is made concrete using the concepts of word frequency and the inverse document frequency defined below.

Definition 18 *Let w_i be a word in document d_j of a collection C. The* term frequency *of w_i in d_j (denoted by tf_{ij}) is equal to the number of occurrences of word w_i in d_j. The* document frequency *of word w_i in the collection C (denoted by df_i) is equal to the number of documents in C that contain w_i. The* inverse document frequency *of w_i is then given by $idf_i = \frac{1}{df_i}$. Finally, the number $tf_{ij} \cdot idf_i$ will be called the* weight *of word w_i in document d_j and will be denoted by ω_{ij}.*

At this point we should stress that the concept of inverse document frequency assumes that there is a *collection* of documents which is used in the calculation. In our dissemination scenario we assume that for each attribute A there is a collection of text values C_A that is used for calculating the *idf* values to be used in similarity computations involving attribute A (the details are given below). C_A can be a collection of recently processed text values as suggested in [145].

We are now ready to define the main new concept in $\mathcal{AWPS}$, the similarity of two text values. The similarity of two text values s_q and s_d is defined as the cosine of the angle formed by their corresponding vectors:[9]

$$sim(s_q, s_d) = \frac{s_q \cdot s_d}{\|s_q\| \cdot \|s_d\|} = \frac{\sum_{i=1}^{N} w_{q_i} \cdot w_{d_i}}{\sqrt{\sum_{i=1}^{N} w_{q_i}^2 \cdot \sum_{i=1}^{N} w_{d_i}^2}} \tag{3.1}$$

By this definition, similarity values are real numbers in the interval $[0, 1]$.

Let us now proceed to give the syntax of the query language for $\mathcal{AWPS}$. Since $\mathcal{AWPS}$ extends $\mathcal{AWP}$, a query in the new model is given by Definition 3.4 with one more case for atomic queries:

[8] Note that in the VSM model and systems adopting it (e.g., WHIRL [34]) word *stems*, produced by some stemming algorithm [108], are forming the vocabulary instead of words. Additionally, *stopwords* (e.g., "the") are eliminated from the vocabulary. These important details have no consequence for the theoretical results of this chapter, but it should be understood that our current implementation of the ideas of this section utilizes these standard techniques.

[9] The IR literature gives us several very closely related ways to define the notions of weight and similarity [10, 91, 141]. All of these weighting schemes come by the name of $tf \cdot idf$ weighting schemes. Generally a weighting scheme is called $tf \cdot idf$ whenever it uses word frequency in a monotonically increasing way, and document frequency in a monotonically decreasing way.

- $A \sim_k s$ where $A \in \mathcal{A}$, s is a text value over $\mathcal{V}$ and k is a real number in the interval $[0, 1]$.

Example 22 *The following are some queries in $\mathcal{AWPS}$ using the schema of Example 17:*

$$BODY \sim_{0.6} \text{“}Milos\ is\ the\ ideal\ place\ for\ holidays\ by\ the\ beach\text{”},$$
$$(SENDER \sqsupseteq (John \prec_{[0,2]} Brown)) \wedge$$
$$(TITLE \sim_{0.9} \text{“}Hotels\ and\ resorts\ in\ Greece\text{”}),$$
$$BODY \sim_{0.9} \text{“}Stock\ options\ during\ Easter\ holidays\text{”}$$

We now give the semantics of our query language, by defining when a document satisfies a query. Naturally, the definition of satisfaction in $\mathcal{AWPS}$ is as in Definition 14 with one additional case for the similarity operator:

- If ϕ is of the form $\mathcal{A} \sim_k s_q$ then $d \models \phi$ iff there exists a pair $(A, s_d) \in d$ and $sim(s_q, s_d) \geq k$.

The reader should notice that the number k in a similarity predicate $A \sim_k s$ gives a *relevance threshold* that candidate text values s should exceed in order to satisfy the predicate. This notion of relevance threshold was first proposed in an information dissemination setting by [50] and later on adopted by [145]. The reader is asked to contrast this situation with the typical information retrieval setting where a ranked list of documents is returned as an answer to a user query. This is not a relevant scenario in an information dissemination system because very few documents (or even a single one) enter the system at a time, and need to be forwarded to interested users.

A low similarity threshold in a predicate $A \sim_k s$ might result in many irrelevant documents satisfying a query, whereas a high similarity threshold would result in very few achieving satisfaction (or even no documents at all). In an implementation of our ideas, users can start with a certain relevance threshold and then update it using relevance feedback techniques to achieve a better satisfaction of their information needs. Recent techniques from adaptive IR can be utilised here [23].

Example 23 *The first query of Example 22 is likely to be satisfied by the document of Example 17 (of course, we cannot say for sure until the exact weights are calculated in the manner suggested above). The second query is not satisfied, since attribute $TITLE$ does not exist in the document. Moreover the third query is unlikely to be satisfied since the only common word between the query and Example 17 is the word “holiday”.*

3.6 Conclusions

In this chapter we concentrated on data models and query languages for textual information dissemination. We concentrated on defining the syntax and semantics of three progressively more extensive models and languages, namely $\mathcal{WP}$, $\mathcal{AWP}$, $\mathcal{AWPS}$.

Data model $\mathcal{WP}$ is based on free text and its corresponding query language on the boolean model with proximity operators. Moreover data model $\mathcal{AWP}$ is based on attributes with finite-length strings as values.Its query language is an extension of the query language of data model $\mathcal{WP}$. Finally $\mathcal{AWPS}$ extends $\mathcal{AWP}$ by introducing a similarity operator in the style of modern IR. In the next chapter we define four important reasoning problems that arise when the models of this section are used for information dissemination scenarios such as the ones surveyed in Chapter 2 and study their computational complexity.

Chapter 4

The Complexity of Information Dissemination with $\mathcal{WP}$ and $\mathcal{AWP}$

The previous chapter presented the textual information models $\mathcal{WP}$, $\mathcal{AWP}$ and $\mathcal{AWPS}$. In this chapter we study the computational complexity of the following problems that arise when the models $\mathcal{WP}$ and $\mathcal{AWP}$ are used in information dissemination environments:

1. *Satisfiability:* Given a query q, is it satisfiable?
2. *Satisfaction:* Given a document d and a query q, does d satisfy q?
3. *Filtering:* Given a database of long-standing queries db and an incoming document d, which elements of db satisfy q?
4. *Entailment* (equivalently, *subsumption*): Given two queries q_1 and q_2, is it the case that q_1 entails q_2 (equivalently, q_2 subsumes q_1)?

It should be apparent that finding efficient algorithms for the solution of the above problems is an important requirement in the development of textual information dissemination systems. Our results are summarized in the following table:

Queries / **Problem**	$\mathcal{WP}$	$\mathcal{WP}$-DNF	$\mathcal{AWP}$	$\mathcal{AWP}$-DNF
Satisfaction	PTIME	PTIME	PTIME	PTIME
Matching	PTIME	PTIME	PTIME	PTIME
Satisfiability	NP-complete	PTIME	NP-complete	PTIME
Entailment	coNP-complete		coNP-complete	

The notation $\mathcal{WP}$-DNF or $\mathcal{AWP}$-DNF means that we are referring to the class of queries in $\mathcal{WP}$ or $\mathcal{AWP}$ that are in DNF form.

From the above table we see that the problems of satisfaction and matching can always be solved in PTIME for both models of this chapter.

Unfortunately, things do not look as bright for the satisfiability problem: if one wants a PTIME algorithm for these problems, he has to concentrate on special classes of queries (for example, queries in DNF form). The remaining sections of this chapter present our complexity results in more detail. Things are similarly hard for the subsumption problem.

4.1 The Complexity of Satisfaction and Filtering

In this section we present PTIME upper bounds for the satisfaction problem and filtering problem in models $\mathcal{WP}$ and $\mathcal{AWP}$.

In previous research, [28] have presented a method for evaluating positive word patterns with proximity operators kW and kN on sets of text values (Fig. 2 of [28]). This method is intended to provide semantics to word patterns, and nothing is said about the computational complexity of evaluation. In this chapter we have followed the more formal route of *separating* the definition of semantics from the algorithms and complexity of deciding satisfaction.

We start with the satisfaction problem for proximity-free word patterns of the model $\mathcal{WP}$.

Lemma 1 *Let s be a text value and wp a proximity-free word pattern. We can decide whether $s \models wp$ in $O(\delta + \rho)$ time on average where δ is the number of words and ρ is the number of operators in wp.*

Proof: We can easily construct a recursive algorithm that operates on a parse tree representation of wp and solves the problem. As it is common in IR research [10], each text value s will be represented by a hash table of size m with hash function h that maps each word $w \in \mathcal{V}$ to an element of $\{0, \ldots, m-1\}$. Each slot of the hash table can be a linked list of words that hash to this slot (i.e., collisions are handled by chaining [35]). We also assume that the computation of the hash function takes $O(1)$ time, and that the average number of elements stored in each chain is constant. Then, at each leaf of the parse tree corresponding to a word w, our algorithm will decide $s \models w$ in $O(1)$ time *on average* using the hash table for s. At inner nodes of the parse tree, a single yes/no value will suffice for the completion of the algorithm. ∎

We now turn to proximity word patterns. We first need the following lemma.

Lemma 2 *Let s be a text value and wp a positive proximity-free word pattern. Function eval-pos-prox-free(wp, s) shown in Figure 4.1 returns a non-empty set*

function *eval-pos-prox-free*(wp, s)
if wp is a word of $\mathcal{V}$ **then**
  return $\{\ [x,x] :\ s(x) = wp\ \}$
else if wp is of the form $wp_1 \wedge wp_2$ **then**
  return $\{\ [min(l_1,l_2), max(u_1,u_2)] :\ [l_1,u_1] \in$ *eval-pos-prox-free*(wp_1,s) and
    $[l_2,u_2] \in$ *eval-pos-prox-free*$(wp_2,s)\ \}$
else if wp is of the form $wp_1 \vee wp_2$ **then**
  return $\{\ [l,u] :\ [l,u] \in$ *eval-pos-prox-free*$(wp_1,s)\ \cup$ *eval-pos-prox-free*$(wp_2,s)\ \}$
else if wp is of the form (wp_1) **then**
  return *eval-pos-prox-free*(wp_1)

function *prox-comp*(wp, s)
if wp is a positive proximity-free word pattern **then**
  return *eval-pos-prox-free*(wp, s)
else
  Let wp be $wp_1 \prec_i rest$ where $rest$ is a proximity word pattern
  return $\{\ [l_1,u_1] :\ [l_1,u_1] \in$ *eval-pos-prox-free*(wp_1,s) and there exists a position
    interval $[l_2,u_2] \in$ *prox-comp*$(rest,s)$ such that $l_2 - u_1 - 1 \in i\ \}$
end

Figure 4.1: Some useful functions for deciding whether $s \models wp$

of position intervals O iff $s \models wp$. Additionally, for every set of positions P such that $s \models_P wp$ there exists an interval $[l,u] \in O$ such that

$$P \subseteq [l,u],\ min(P) = l \text{ and } max(P) = u.$$

The set O can be computed in $O((\delta+\rho)\,|s|^4)$ time where δ is the number of words and ρ is the number of operators in wp.

Proof: Let us first prove the "only-if" part of the first statement of the lemma i.e., if the call *eval-pos-prox-free*(wp,s) returns a non-empty set then $s \models wp$. The proof is by induction on the structure of wp.

Base case: wp is a word w. If the call *eval-pos-prox-free*(wp,s) returns a non-empty set, then the first if-statement of the function returns a non-empty set. In other words, there exists $x \in \{1,\ldots,|s|\}$ and $s(x) = wp$. Thus $s \models wp$.

Inductive step: Let us first consider the case when wp is of the form $wp_1 \wedge wp_2$. If the call *eval-pos-prox-free*(wp,s) returns a non-empty set, then both calls *eval-pos-prox-free*(wp_1,s) and *eval-pos-prox-free*(wp_2,s) return non-empty sets. The inductive hypothesis then implies that $s \models wp_1$ and $s \models wp_2$. Therefore $s \models wp$. The proof is similar for the other two cases of the inductive step.

Let us now consider the"if" part. It is easy to see that the "if" part follows immediately from the second statement, so let us consider that second statement instead. The proof is again by induction on the structure of wp.

Base case: wp is a word and the lemma follows immediately (see the first if statement of function *eval-pos-prox-free*).

Inductive step: Let us first consider the case when wp is of the form $wp_1 \wedge wp_2$. Let P be a set of positions such that $s \models_P wp$. Then Definition 20 implies that there exist sets of positions $P_1, P_2 \subseteq \{1, \ldots, |s|\}$ such that $s \models_{P_1} wp_1$, $s \models_{P_2} wp_2$ and $P = P_1 \cup P_2$. Let O_1 and O_2 be the sets returned by the recursive calls *eval-pos-prox-free*(wp_1, s) and *eval-pos-prox-free*(wp_2, s). The inductive hypothesis now implies that there exist position intervals $[l_1, u_1] \in O_1$ and $[l_2, u_2] \in O_2$ such that

$$P_1 \subseteq [l_1, u_1],\ min(P_1) = l_1 \text{ and } max(P_1) = u_1$$

and

$$P_2 \subseteq [l_2, u_2],\ min(P_2) = l_2 \text{ and } max(P_2) = u_2.$$

It is easy to see now that

$$P \subseteq [min(l_1, l_2), max(u_1, u_2)],\ min(P) = min(l_1, l_2) \text{ and } max(P) = max(u_1, u_2).$$

Thus the conclusion of the lemma holds for wp. The proof is similar for the other two cases of the inductive step.

Let us now calculate the complexity of *eval-pos-prox-free* assuming that we represent word pattern wp by a parse tree and text value s by a hash table as in Lemma 1. Now each slot of the hash table will be a linked list of all pairs (w, x) where w is a word of s that hashes to this slot and x is a position of w in s. At each leaf of the parse tree corresponding to a word w, the function calculates the set of all position intervals $[x, x]$ such that $s(x) = w$. Each such set is of size $O(|s|)$ and can be computed in $O(|s|)$ time (under assumptions about hashing similar to the ones in Lemma 1). At each inner node of the parse tree corresponding to an operator $\wedge, \vee$ or $\neg$, the sets of position intervals computed by the function are of size at most $O(|s|^2)$ and can be computed in at most $O(|s|^4)$ time. Thus, the set returned finally by *eval-pos-prox-free* is of size $O(|s|^2)$ and is computed in $O((\delta + \rho)\,|s|^4)$ time.

■

We now use the above lemma to compute an upper bound on the complexity of satisfaction for proximity word patterns.

Lemma 3 *Let s be a text value and wp a proximity word pattern. We can decide whether $s \models wp$ in $O(n(\delta_{max}+\rho_{max})\,|s|^4)$ time where n is the number of proximity free subformulas of wp, δ_{max} is the maximum number of words in a proximity-free subformula of wp and ρ_{max} is the maximum number of operators in a proximity-free subformula of wp.*

Proof: The recursive function *prox-comp* shown in Figure 4.1 can be used for the required computation. Let the input wp be of the form

$$wp_1 \prec_{i_1} \cdots \prec_{i_{n-1}} wp_n$$

where $wp_1, wp_2, \ldots, wp_n$ are positive proximity-free word patterns. Then s satisfies wp if and only if *prox-comp* returns a non-empty set.

prox-comp recursively reaches the last proximity-free word pattern wp_n and with a call to *eval-pos-prox-free* computes and returns the set of all position intervals $[l, u]$ that certify satisfaction of wp_n. Let us denote this set by O. This is done by the if-part of the function. Then *prox-comp* computes all position intervals $[l, u]$ that certify satisfaction of wp_{n-1} and keeps only the subset of these that are in the required distance with some element of O. This subset is returned so that it can play the role of set O in a similar computation involving wp_{n-2}. This is done in the else-part of the function. In this way, when we reach the first proximity-free word pattern wp_1, a chain of position intervals has been established that verifies that $s \models wp$. If at any point an empty set is returned, this set is propagated back and obviously $s \not\models wp$. A formal proof can be easily done by induction and is omitted.

For the complexity bound notice that according to Lemma 2, each call to *eval-prox-free* can be done in $O((\delta_{max} + \rho_{max})\,|s|^4)$ time and returns a set of size $O(|s|^2)$. Thus the worst-case complexity of *prox-comp* is $O(n(\delta_{max} + \rho_{max})\,|s|^4)$. ∎

We can now show that satisfaction can be decided in PTIME for all formulas in $\mathcal{WP}$.

Theorem 1 *Let s be a text value and wp a word pattern. The problem of deciding whether $s \models wp$ can be solved in $O(\mu n_{max}(\delta_{max} + \rho_{max})\,|s|^4)$ time where μ is the number of operators $\wedge$ and $\vee$ in wp, n_{max} is the maximum number of proximity-free subformulas in a proximity word pattern of wp, δ_{max} is the maximum number of words in a proximity-free subformula of wp and ρ_{max} is the maximum number of operators in a proximity-free subformula of wp.*

Proof: Given the functions *eval-pos-prox-free* and *prox-comp* of the above lemmas, it is easy to write a recursive function that decides $s \models wp$ by traversing a parse tree for wp (as implied by Definition 4). In this case, we do not need to keep track of sets of position intervals at inner nodes of the parse tree corresponding to proximity-free word patterns (Case 1 of Definition 4), proximity word patterns (Case 2 of Definition 4) and operators $\wedge$ and $\vee$ (Cases 3 and 4 of Definition 4). In fact, a single yes/no value will suffice. The correctness follows easily by an inductive argument. The complexity bound also follows easily from the above lemmas. ∎

We close this section by showing that satisfaction can be decided in PTIME for all formulas of $\mathcal{AWP}$ as well.

Theorem 2 *Let d be a document and ϕ be a query in $\mathcal{AWP}$. Deciding whether $d \models \phi$ can be done in $O((E + H)(\alpha + V)MN(\Delta + P)^2 S^4)$ time where E is the number of atomic subqueries in ϕ, H is the number of operators in ϕ, α is the number of attributes in d, V is the maximum size of a text value appearing in ϕ,*

M is the maximum number of operators $\wedge$ and $\vee$ in a word pattern of ϕ, N is the maximum number of proximity-free subformulas in a proximity word pattern of ϕ, Δ is the maximum number of words in a proximity-free subformula of ϕ, P is the maximum number of operators in a proximity-free subformula of ϕ, and S is the maximum size of a text value appearing in d.

Proof: We assume that ϕ is represented by a parse tree. We also assume that d is represented by a linked list of triples (A, s, T_s) where A is an attribute, s is a text value and T_s is a hash table mapping each word of s to its position in s. Thus, we have a dual representation for text value s. The former representation will be used for evaluating queries of the form $A = s$ and the latter for queries $A \sqsupset wp$.

Evaluating a subquery of ϕ of the form $A = v$ can be done in time $O(\alpha + |v|)$ where α is the number of attributes in d and $|v|$ is the size of text value v.

Using Theorem 1, we can see that evaluating a subquery of the form $A \sqsupset wp$ can be done in time $O(\alpha + n_{max}\mu(\delta_{max} + \rho_{max})\,|s|^4)$ where α is the number of attributes in d and the rest of the parameters are as in Theorem 1.

Given the above bounds, we can easily decide whether $d \models \phi$ by a recursive algorithm that traverses the parse tree for ϕ. This algorithm evaluates atomic subqueries at the leaves while at intermediate nodes (corresponding to operators) computes a yes/no value. The complexity bound follows easily now. ∎

To solve the filtering problem in models $\mathcal{WP}$ and $\mathcal{AWP}$ one should solve $|db|$ satisfaction problems where $|db|$ is the size of the database of long-standing queries db. Thus the filtering problem can also be solved in PTIME (the exact upper bounds are omitted because they can be easily computed using Theorems 1 and 2). In practice one would create *indices* over the database of long-standing queries db to solve the filtering problem more efficiently [143, 145]. This is the approach we take in Chapter 5 of this report.

4.2 Satisfiability and Entailment in $\mathcal{WP}$

We now turn our attention to the satisfiability and entailment problems for queries in $\mathcal{WP}$. Let the satisfiability problem for proximity-free word patterns be denoted by PFWP-SAT. There is an obvious connection of PFWP-SAT and SAT, the satisfiability problem for Boolean logic [102]. Any instance of PFWP-SAT can be considered to be an instance of SAT and vice versa (this is a trivial reduction where the roles of words and Boolean variables are interchanged). Thus we only have to consider the complications arising in our framework due to proximity word patterns.

In what follows, we will need the binary operation of *concatenation* of two text values.

Definition 19 *Let s_1 and s_2 be text values over vocabulary $\mathcal{V}$. Then the* concatenation *of s_1 and s_2 is a new text value denoted by $s_1 s_2$ and defined by the following:*

1. $|s_1 s_2| = |s_1| + |s_2|$
2. $s_1 s_2(x) = s_1(x)$ *for all* $x \in \{1, \ldots, |s_1|\}$, *and*
3. $s_1 s_2(x) = s_2(x)$ *for all* $x \in \{|s_1| + 1, \ldots, |s_2| + |s_1|\}$

We will also need the concept of the *empty text value* which is denoted by ϵ and has the property $|\epsilon| = 0$. The following properties of concatenation are easily seen:

1. $(s_1 s_2)s_3 = s_1(s_2 s_3)$, for all text values s_1, s_2 and s_3.
2. $s\epsilon = \epsilon s = s$ for every text value s.

The associativity of concatenation allows us to write concatenations of more than two text values without using parentheses.

The following variant of the concept of satisfaction captures the notion of a set of positions in a text value containing *only* words that contribute to the satisfaction of a proximity-free word pattern. This concept is used in the results that follow.

Definition 20 *Let $\mathcal{V}$ be a vocabulary, s a text value over $\mathcal{V}$, wp a proximity-free word pattern over $\mathcal{V}$, and P a subset of $\{1, \ldots, |s|\}$. The concept of s* satisfying *wp* with set of positions *P (denoted by $s \models_P wp$) is defined as follows:*

1. *If wp is a word of $\mathcal{V}$ then $s \models_P wp$ iff there exists $x \in \{1, \ldots, |s|\}$ such that $P = \{x\}$ and $s(x) = wp$.*
2. *If wp is of the form $wp_1 \wedge wp_2$ then $s \models_P wp$ iff there exist sets of positions $P_1, P_2 \subseteq \{1, \ldots, |s|\}$ such that $s \models_{P_1} wp_1$, $s \models_{P_2} wp_2$ and $P = P_1 \cup P_2$.*
3. *If wp is of the form $wp_1 \vee wp_2$ then $s \models_P wp$ iff $s \models_P wp_1$ or $s \models_P wp_2$.*
4. *If wp is of the form (wp_1) then $s \models_P wp$ iff $s \models_P wp_1$.*

We also need the following notation. Let P be a subset of the set of natural numbers $\mathbb{N}$, and $x \in \mathbb{N}$. We will use the notation $P + x$ to denote the set of natural numbers $\{p + x : \ p \in P\}$.

The following lemma is now easy to see.

Lemma 4 *Let s and s' be text values, wp a proximity-free word pattern and $P \subseteq \{1, \ldots, |s|\}$. If $s \models_P wp$ then $ss' \models_P wp$ and $s's \models_{P+|s'|} wp$.*

The following proposition shows that positive proximity-free word patterns are always satisfiable (its proof can be done by induction on the structure of the word pattern).

Proposition 11 *If wp is a positive proximity-free word pattern then wp is satisfiable. In fact, there exists a text value s_0 such that*

1. *$|s_0| \leq words(wp) \cdot ops(wp)$ where $words(wp)$ is the number of words of wp (multiple occurrences of the same word are multiply counted) and $ops(wp)$ is the number of operators of wp (or 1 if wp has no operators).*
2. *Every word of s_0 is a word of wp.*
3. *$s_0 \models_{\{1,\ldots,|s_0|\}} wp$.*

We can easily show that proximity word patterns are also always satisfiable.

Proposition 12 *Let wp be a proximity word-pattern of the form*

$$w_1 \prec_{i_1} \cdots \prec_{i_{n-1}} w_n.$$

Then wp is satisfied by a text value $s = w_1 v_1 \cdots v_{n-1} w_n$ where v_k, $k = 1, \ldots, n-1$ are text values of the following form: If $begin(i_k) > 0$ then v_k is formed by $begin(i_k)$ successive occurrences of the special word # which is not contained in wp. Otherwise, v_k is the empty text value ϵ.

Finally, we can show that any positive word pattern is always satisfiable.

Proposition 13 *Let wp be a positive word pattern and $\theta_1 \vee \cdots \vee \theta_k$ be the DNF of wp. Then there exists a $j \in \{1, \ldots, k\}$, and text values $s_{0j}, s_{1j}, \ldots, s_{mj}$ such that $s_{0j} s_{1j} \cdots s_{mj} \models wp$ and*

1. *s_{0j} is a sequence of words appearing as conjuncts of disjunct θ_j, and*
2. *for $i = 1, \ldots, m$, s_{ij} is a text value such that $s_{ij} \models_{\{1,\ldots,|s_{ij}|\}} \phi_i$ where $\phi_1, \ldots, \phi_m$ are all the proximity conjuncts of θ_j.*

The next theorem shows that when negation is introduced, deciding the satisfiability of a word pattern becomes a hard computational problem. But first we need a lemma that shows that even in the case of arbitrary word patterns, satisfiability implies satisfaction by a text value of a special form.

Lemma 5 *Let wp be a word pattern and $\theta_1 \vee \cdots \vee \theta_k$ be the DNF of wp. Then wp is satisfiable iff there exists $j \in \{1, \ldots, k\}$, and text values $s_{0j}, s_{1j}, \ldots, s_{mj}$ such that $s_{0j} s_{1j} \cdots s_{mj} \models wp$ and*

1. *s_{0j} is a sequence of words appearing non-negated in disjunct θ_j, and*

2. *for* $i = 1, \ldots, m$, s_{ij} *is a text value such that* $s_{ij} \models_{\{1,\ldots,|s_{ij}|\}} \phi_i$ *where* $\phi_1, \ldots, \phi_m$ *are all the proximity conjuncts of* θ_j.

We can now utilize Lemma 5 to show the upper bound in the following result (the lower bound is easy).

Theorem 3 *Let wp be a word pattern. Deciding whether wp is satisfiable is an NP-complete problem.*

A tractable case of the satisfiability problem for word patterns in $\mathcal{WP}$ is given by the following easy theorem.

Theorem 4 *Let wp be a word pattern in DNF. Deciding whether wp is satisfiable can be done in* $O(\kappa^2\zeta)$ *time where* κ *is the maximum number of words in a conjunct of wp, and* ζ *is the number of conjuncts.*

The following proposition gives the usual relation between entailment and unsatisfiability of Boolean logic as it should be stated in our framework. Note that wp_2 is required to be proximity-free so that the negation operator can be applied.

Proposition 14 *Let* wp_1 *and* wp_2 *be word patterns and* wp_2 *is proximity-free.* $wp_1 \models wp_2$ *iff* $wp_1 \wedge \neg wp_2$ *is unsatisfiable.*

Let us now turn our attention to the entailment problem in $\mathcal{WP}$. The problem is easily seen to be coNP-hard from the previous Proposition. Using techniques similar to the ones developed above we can show that to decide whether $\phi \models \psi$ for word patterns ϕ and ψ it is enough to consider text values of a special form. This allows to prove that the entailment problem is in coNP thus we have the following result.

Theorem 5 *Deciding whether a word pattern in* $\mathcal{WP}$ *entails another is a coNP-complete problem.*

4.3 Satisfiability and Entailment in $\mathcal{AWP}$

Let us now turn our attention to the satisfiability and entailment problems for queries in $\mathcal{AWP}$. Let $\mathcal{Q}$ denote the class of queries in $\mathcal{AWP}$ with the following property: every positive word pattern *wp* appearing in formulas of the form $A \sqsupseteq wp$ is in DNF or CNF form. Let SAT($\mathcal{Q}$) denote the satisfiability problem for queries in class $\mathcal{Q}$. The following two propositions show that the problems SAT and SAT($\mathcal{Q}$) are equivalent under polynomial time reductions.

Proposition 15 SAT is polynomially reducible to SAT($\mathcal{Q}$).

Proposition 16 SAT($\mathcal{Q}$) is polynomially reducible to SAT.

Proof: Let ϕ be a query in $\mathcal{Q}$. Using Proposition 9, ϕ can easily be transformed into a formula θ which is a Boolean combination of atomic queries (see Definition 17) using only operators $\wedge$ and $\vee$. This transformation can be done in time linear in the size of the formula.

The next step is to substitute in θ atomic formulas $A = s$ and $A \sqsupseteq wp$ (where wp is a word or a proximity word pattern) by propositional variables $p_{A=s}$ and $p_{A \sqsupseteq wp}$ respectively to obtain formula θ'. Finally, the following formulas are conjoined to θ' to obtain ψ:

1. If $A = s_1$ and $A = s_2$ are conjuncts of θ' and $s_1 \neq s_2$ then conjoin $p_{A=s_1} \equiv \neg p_{A=s_2}$.
2. If $A = s$ and $A \sqsupseteq wp$ are conjuncts of θ' and $s \models wp$ then conjoin $p_{A=s} \supset p_{A \sqsupseteq wp}$.
3. If $A = s$ and $A \sqsupseteq wp$ are conjuncts of θ' and $s \not\models wp$ then conjoin $p_{A=s} \supset \neg p_{A \sqsupseteq wp}$.
4. If $A \sqsupseteq wp_1$ and $A \sqsupseteq wp_2$ are conjuncts of θ' and $wp_1 \models wp_2$ then conjoin $p_{A \sqsupseteq wp_1} \supset p_{A \sqsupseteq wp_2}$.

The above step can be done in polynomial time because satisfaction and entailment of word patterns in θ' can be done in polynomial time.

It is also easy to see that ϕ is a satisfiable query iff ψ is a satisfiable formula of Boolean logic. Then the result holds. ∎

Thus we have the following result.

Theorem 6 *The problem SAT($\mathcal{Q}$) is NP-complete.*

Given the reduction of Proposition 16, one can discover tractable subcases of the problem SAT($\mathcal{Q}$). As an example, an easy reduction from 2-SAT gives us the following result.

Corollary 1 *Let ϕ be a query of $\mathcal{AWP}$ such that each disjunction of ϕ has at most two disjuncts. The problem of deciding whether ϕ is satisfiable can be solved in PTIME.*

Finally, what is important about Proposition 16 is that it gives us a straightforward way of evaluating the satisfiability of a query ϕ by transforming it into a propositional formula ψ and invoking a well-known propositional satisfiability algorithm on ψ (e.g., variations of GSAT [114]). One could also devise filtering algorithms that are based on Boolean logic techniques as it is done for a similar language in [19].

The following theorem can now be proved using similar ideas.

Theorem 7 *The entailment problem for queries of $\mathcal{AWP}$ in class $\mathcal{Q}$ is coNP-complete.*

4.4 Conclusions

In this section we have studied the following four fundamental problems that arise in textual information dissemination systems like the ones presented in Chapter 2:

1. *Satisfiability.* Deciding whether a given query ϕ can be satisfied by any document at all.

2. *Satisfaction or matching.* Deciding whether an incoming document satisfies (or matches) a query ϕ.

3. *Filtering.* Given a database of profiles db and an incoming document d, find all profiles $q \in db$ that match d.

4. *Entailment or subsumption.* Deciding whether a profile is more or less "general" than another.

In the next section we concentrate our attention on the filtering problem as it arises in textual information dissemination and study efficient main memory algorithms.

Chapter 5

Fast Algorithms for Textual Information Dissemination under the Model $\mathcal{AWP}$

In this chapter we present two main memory algorithms suitable for the filtering problem in textual information dissemination systems. The algorithms presented are designed for a subset of the model $\mathcal{AWP}$. The model $\mathcal{AWP}$ presented in detail in Section 3.4 and also in [69, 70, 73, 41]. We design and carry out a realistic evaluation of these algorithms based on a special corpus of documents originally compiled in [40], and later on used in [74] for the evaluation of various filtering algorithms for the special case of the model $\mathcal{WP}$. We process the corpus documents to fit for our evaluation and develop a formal way to create realistic profiles following the original methodology of [74].

The rest of this chapter is organised as follows. Section 5.1 describes the algorithms, while Section 5.2 introduces the corpus used and discusses the processing carried out to fit the documents to our needs. Moreover it presents our method for the generation of realistic user profiles. Section 5.3 describes the experimental evaluation carried out. Finally Section 5.4 concludes this chapter.

5.1 Filtering Algorithms for Model $\mathcal{AWP}$

In this section we present two algorithms that solve the problem of filtering for a subset of the model $\mathcal{AWP}$ [69, 70, 73] defined as follows. The concept of document in $\mathcal{AWP}$ will be as it has been defined in Definition 12 of Chapter 3. The concept of query or profile is restricted as follows. A profile can be of the form $A_1 \sqsupseteq wp_1 \ \wedge \ ... \ \wedge \ A_n \sqsupseteq wp_n$, where A_i is an attribute and wp_i is a word pattern containing conjunctions of words and proximity formulas with only words as subformulas. The first approach adopts the brute force strategy and was implemented for comparison purposes. The second approach uses inverted

indexes [10] for accessing the user profiles in an efficient way.

5.1.1 Brute Force Algorithm

The Brute Force algorithm (BF) is a very simple one and was implemented for comparison purposes. BF maintains a linked list where all the profiles are stored and each time a new profile arrives it is inserted at the end of this list.

Document Data Structures

To facilitate the matching process, BF uses a hash table to represent the incoming document. This hash table is called the *occurrence table* (OT), and uses words as keys. An example of an OT is given in Figure 5.2. OT is used for storing all the attributes of the document in which a specific word appears, along with the positions this word occupies in the text that is the value of this attribute. Open addressing, with linear probing is used for the implementation of the hash table [35]. Moreover the size of the hash table is set to a prime number, causing a more uniform distribution of data into the hash table [5, page 435]. This results in lower insertion and search times. This design choice is made for all the instances of the hash tables used in the algorithms described in this section.

Auxiliary Data Structures

Apart from the linked list for storing the profiles and the OT for storing the document, BF uses the following data structures:

1. A linked list, called *Success List* (SL), used for storing the profile identifiers of all the profiles that are found to match the incoming document.

2. An array, called the *Positions Array* (PA), that is used during the evaluation of a proximity formula contained in a profile's attribute. This array stores all the operands (words) of a proximity formula, along with a pointer to the positions of this operand (remember that this information is stored in the OT). This array is used by function *EvAllProx()* which is shown in Figure 5.1.

Matching Process

BF examines the profiles stored in the linked list exhaustively using a fail-first strategy. For every word $w_{i,j}$ contained in an attribute A_j of a profile P, BF probes OT to determine if it is also contained in the corresponding document attribute, stopping as soon as a non-matching word is found. If all words in A_j are found in the document's attribute, BF calls $EvAllProx()$ to determine if the proximity operations are also satisfied. This is repeated for all profile attributes in

```
function EvAllProx(attribute A) {
        for each proximity formula pf_j ∈ A with words w_i do
                if pf_{j-1} MATCHED then
                   call EvOneProx(positions(w_1), positions(w_2))
                else return NOT SATISFIED
        end for
return SATISFIED
}

function EvOneProx(positions(w_i) , positions(w_{i+1})) {
        for each position p_i ∈ w_i do
                for each position p_{i+1} ∈ w_{i+1} do
                        if p_{i+1} − p_i − 1 < distance then
                           if w_{i+1} is the last word in formula then
                              return MATCH
                           else add p_{i+1} in the Candidate_Positions_List_{i+1}
                           end if
                        end if
                end for
        end for
        if w_{i+1} is the last word in formula then
                return NOT MATCH
        else call EvOneProx(Candidate_Positions_List_{i+1}, positions(w_{i+2}))
        end if
}
```

Figure 5.1: Pseudo-code for the proximity evaluation algorithm

profile P. In this way the linked list containing the profiles is scanned sequentially until all profiles have been evaluated.

5.1.2 The Algorithm PIndex

The algorithm PIndex (Profile Indexing) utilises a two-level index over profiles, where the first level corresponds to *attributes* and the second to *words* contained in attribute values. To match an incoming document against a set of profiles, PIndex needs several data structures. Some of these data structures are used for indexing the profiles, some others for representing the incoming document, and there are also auxiliary data structures that are used in order to improve performance, or to facilitate the matching process.

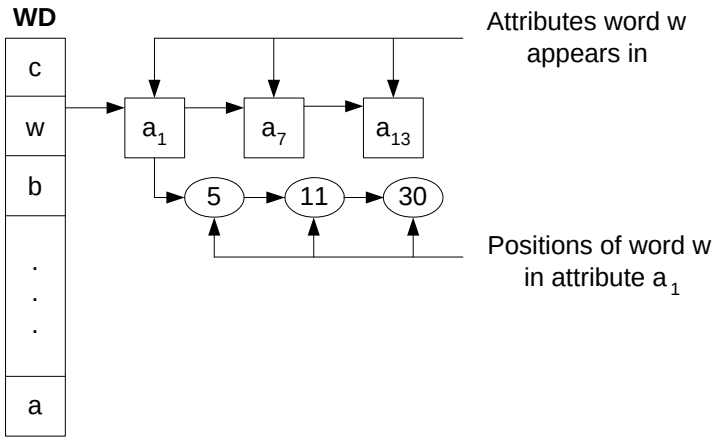

Figure 5.2: A document's occurrence table

Document Data Structures

For each document d PINDEX uses the following data structures:

1. The *distinct attribute list* $DAL(d)$. This is a linked list that stores all the distinct attributes contained in the document. Each element of the list (representing an attribute), points to another linked list, the *distinct word list* (DWL), containing all the distinct words that appear in this attribute. The size of $DAL(d)$ is equal to the number of attributes contained in d.

2. The *occurrence table* $OT(d)$. This is a hash table that uses words as keys. For each word w, $OT(d)$ stores all the attributes of d with values that contain w, along with w's positions in those values. This position information helps PINDEX decide whether an attribute that contains proximity formulas matches the corresponding attribute of the incoming document or not.

Profile Data Structures

To store the profiles in an efficient way, PINDEX uses the following data structures:

1. An array, called the *attribute directory* (AD), that stores pointers to all word directories (WDs are described below). The size of the AD equals the total number of attributes in the attribute set. If attributes are $a_1, a_2, ..., a_n$ then the first element of AD refers to attribute a_1, the second to a_2 and so on.

2. A hash table, called *word directory* (WD), for each attribute in the attribute set. This table is used for storing all the words that are contained in a

profile's attribute. We use a randomly chosen word as the key that each attribute is hashed with. For each attribute in the attribute set, PINDEX maintains a WD which is used to store the words each profile contains in a certain attribute, along with additional information needed (e.g., the profile identifier, information about the proximity formulas contained, distance values in proximity formulas). Each WD slot corresponds to a different word in the profile vocabulary and only profiles that contain this word (and are obviously hashed with it) are stored in the slot.

The above two data structures that store the profiles form a *two-level index* called the *profile index*. Figure 5.3 shows the profile index for the set of profiles:

$$P_1 : \ (A_5 \sqsupseteq a \wedge b \prec_{[0,3]} f) \wedge (A_8 \sqsupseteq b \prec_{[0,2]} w \prec_{[0,3]} g \wedge a)$$

$$P_5 : \ A_5 \sqsupseteq w \wedge f \wedge m$$

Auxiliary Data Structures

Apart from the data structures for storing profiles and documents, PINDEX uses some auxiliary data structures that are useful for the matching process. These are:

1. Two arrays, named $Total$ and $Count$. These arrays are used to help us decide which of the profiles stored match the incoming document. The size of each array equals the number of profiles that are stored in the profile index. Each profile has an entry, that depends on its unique profile identifier, in both of these arrays. In this way the i-th entry in both arrays belongs to profile with identifier i. Array $Total$ stores the number of attributes that are contained in each profile. Array $Count$ is used for counting how many attributes of a profile match the corresponding attributes of the document. As a result, if a profile's entry in array $Total$ equals its entry in $Count$, then the profile matches, since all of its attributes match those of the document.

2. A linked list, called *success list* (SL), used for storing the profile identifiers of all the profiles that are found to match the incoming document.

Let us now discuss the various steps of PINDEX.

The Insertion Algorithm

The algorithm for inserting a profile into the profile index is as follows. Let us assume that the profile p to be inserted in the PI is as follows:

$$A_1 \sqsupseteq wp_1 \ \wedge \ ... \ \wedge \ A_n \sqsupseteq wp_n$$

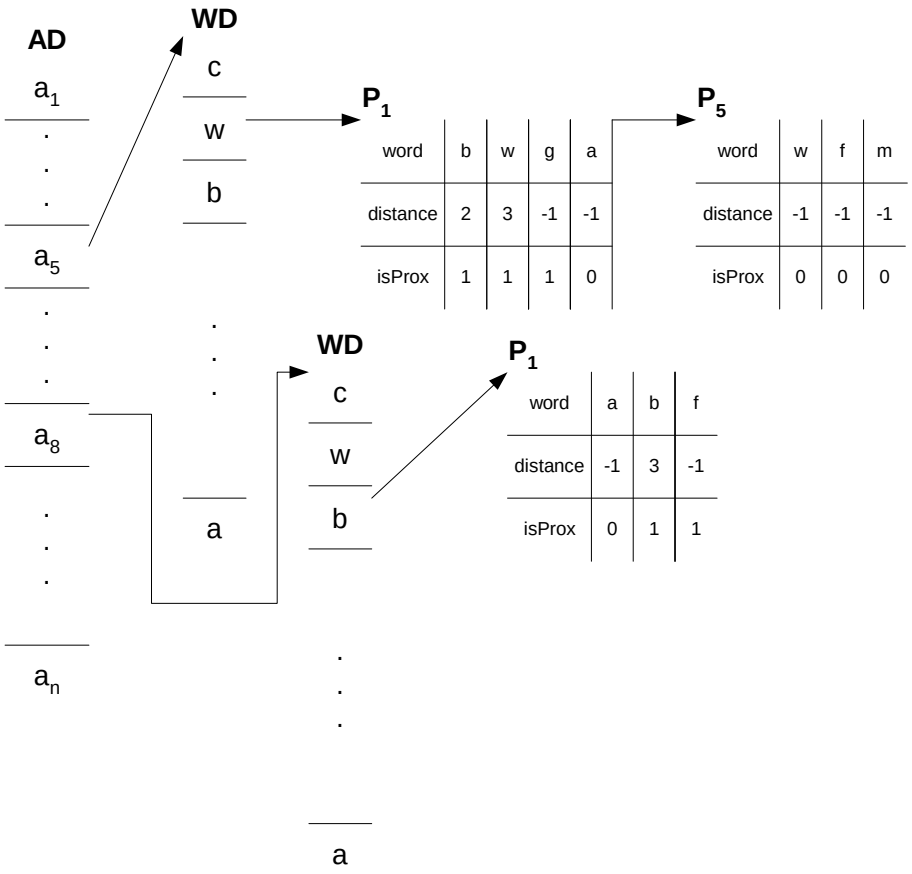

Figure 5.3: The profile index

where A_i is an attribute and wp_i is a word pattern containing conjunctions of words and proximity operations. In the first level of the profile index, p will be hashed according to all attributes A_1, ..., A_n i.e., it will be hashed under n different entries of AD. In the second level, corresponding to an attribute A_i of p, PINDEX chooses a word w of wp_i randomly and hashes p under the entry w of the WD for A_i. All the words forming the attribute formula are stored in a WD slot, along with the word distances that should be satisfied in order a profile to match an incoming document. In order for the algorithm to be able to recognize whether a word is a part of a proximity formula or a simple conjunct, a discrimination bit is used. It follows that no word distances are stored for words that do not participate in a proximity operation. Moreover, additional information useful for the matching process like the profile identifier, is stored.

```
function Match (document d) {
        for each attribute A ∈ DAL(d) do
                for each word w ∈ DWL(A) do
                        probe the AD with A
                        probe the WD pointed to by AD(A) with word w
                        for each profile P_i stored in the WD slot do
                                for each word v ∈ P do
                                        probe OT(d) going to P_{i+1} as
                                        soon as a non-existing word v is found
                                end for
                                call EvAlProx(A) to decide for
                                satisfaction of proximity formulas
                                if all proximity formulas in P_i are satisfied then
                                        Count[i] = Count[i] + 1
                                end if
                        end for
                end for
        end for
        for each entry i of arrays Total and Count
                if Total[i] = Count[i] then
                   add profile identifier to SL(d)
                end if
        end for
}
```

Figure 5.4: Pseudo-code for the matching algorithm

The Filtering Algorithm

The idea behind the filtering algorithm is to examine only those profiles that are possible to match the incoming document, and to do the minimum amount of work needed in order to decide that a profile cannot match the document. The aid to implement our first idea is PI that PINDEX uses for storing the profiles. The aid for our second idea is the observation that proximity queries are similar to conjunctive queries, in the sense that in both types of queries all keywords have to appear in the incoming document, in order for a query to match the document. Taking advantage of this observation, the existence of keywords is checked first and the evaluation of the proximity formulas follows, since it is a time-consuming operation. The pseudo-code for the matching algorithm is given in Figure 5.4. The proximity evaluation algorithm is the same one used by BF and was presented in Figure 5.1.

Description	Value
Number of documents	10,426
Document vocabulary size	641,242
Maximum document size (words)	110,452
Minimum word size (letters)	1
Maximum word size (letters)	35

Table 5.1: Some characteristics of the NN corpus

5.2 Documents and profiles

To evaluate the filtering algorithms presented earlier we use a special document corpus. For the profile generation we use *words* and *technical terms* extracted automatically from the corpus using the method proposed in [52]. In the rest of this section we will introduce our corpus, present the processing that had to be carried out to tailor it to our needs, and show how it can be used to create realistic user profiles. Our intention is to evaluate our algorithms with realistic data that would closely correspond to dissemination scenarios in digital libraries.

5.2.1 The Neural Network Corpus

The corpus we use (called the *NN corpus*) consists of a fraction of research papers from *ResearchIndex* [1, 85] with subject Neural Networks. ResearchIndex, formerly known as Citeseer, is a scientific literature digital library that targets the improvement in the dissemination of scientific literature. ResearchIndex indexes research articles in various formats and provides without charge a variety of useful services. Such services are full-text indexing of the articles enhanced with techniques such as stop words exclusion, autonomous citation indexing, linking the articles with the cited papers and statistics for all articles in the database.

The NN corpus consists of 10,426 scientific papers in English. Some important values for this corpus are summarised in Table 5.1. The NN corpus was initially compiled and processed by Evangelos Milios and his group at Dalhousie University[1]. The documents were downloaded from the ResearchIndex site as postscript files and were converted to text files. Then all references and equations were removed and each word in the document was assigned a grammatical tag (e.g. noun, verb etc.) using a simple rule-based part of speech (POS) tagger [16]. This processing was necessary as a first step for the extraction of multi-word terms by the C-value/NC-value method described in [52].

To use the corpus for our experiments, further processing had to be carried out. Two important components were used for this further processing. The

[1] `http://www.cs.dal.ca/~eem`

Attribute	% fraction of documents
TITLE	63%
AUTHORS	58%
ABSTRACT	88%
BODY	86%
YEAR	63%

Table 5.2: Percentage of documents containing each attribute

first one was the corpus itself in an easily processable form prepared by Thodoros Koutris [74] and the full citation graph of ResearchIndex that was made available to us from . This processing progressed as follows.

Initially we removed all the POS tags from all the documents. We then used the information from the full citation graph[2] of ResearchIndex to extract the title, authors and year of the publication. This information was *not* extracted from the actual corpus since the flat form of the documents contained considerable noise even after several rule-based filters were applied to it. The next step was to process the abstracts that were also provided to us by the Dalhousie group as POS-tagged text files, extracted from the original postscript files. After processing the abstracts we used them in the extraction of the actual body of the documents. After the processing phase was completed, we merged the different attributes extracted, along with the appropriate attribute tags. We then had at our disposal an attribute-tagged corpus with five fields: TITLE, AUTHORS, ABSTRACT, BODY and YEAR. At this point we have to stress out that the information obtained from the citation graph was incomplete, resulting in documents without all the attribute fields filled in. This is actually not a problem in our experimental setting since in an information dissemination scenario users may post documents with only some of the attributes filled in. Table 5.2 gives some interesting measures of the fraction of documents out of the document corpus that contain each attribute, whereas Table 5.3 summarises the fraction of documents that contain a specific number of attributes.

Attribute YEAR will not be used in our evaluation because it does not appear to be useful for expressing user profiles in our target application.

5.2.2 Automatic Term Extraction

The multi-word terms used in the profiles for our experiments are extracted from the NN corpus using the C-value/NC-value approach of [52]. The process of identification of *terms* or *technical terms* or *terminological phrases* from a collection of documents belongs to the research area called *automatic term recog-*

[2]The citation graph was also made available to us by Evangelos Milios[7].

Number of attributes	% fraction of documents
1	7.4%
2	28%
3	1.9%
4	16%
5	45%

Table 5.3: Percentage of documents containing one or more attributes

nition. The C-value/NC-value approach of [52] specifies the "termhood" of a candidate multi-word term as the probability to be a real term. C-value stands for *Collocation-value*, whereas NC-value stands for *New Collocation value*. The C-value of a term is an enhancement of the common statistical measure of frequency of occurrence, incorporating information about nested terms, whereas NC-value embodies information form words that appear in the vicinity of terms in texts. Both methods have been shown to perform better than the classical frequency of occurrence measure in terms of precision and recall [40]. For details on the C-value/NC-value method the reader is invited to see [52, 40, 74].

5.2.3 Profile Generation

The main construct in our profile creation process is that of a *unit*. Units in our context represent different entities that can be used to create a profile. In the experiments that will be described in Section 5.3 we used four different sets of units for the generation of the profiles database. The first two unit sets consist of proximity formulas created from multi-word terms, that were extracted from the NN corpus using the NC-value method described earlier. The third one is the set of all the nouns extracted from document abstracts, and the fourth one is the set of all author last names in the NN corpus documents. Combining units from these four sets in a well defined way allows us to create realistic profiles databases in order to conduct our experiments.

Creation of the Different Unit Sets

The creation of the first two unit sets was based on the extraction of multi-word terms from the corpus. To create these sets, a ranked list of multi-word terms was extracted from the corpus documents. We then excluded from this list all terms that contained more than five words since they were noise produced by the C-/NC-value methods. Additionally we specified an upper and lower NC-value cut-off threshold for the terms remaining in the list. These cut-off thresholds were used to increase the discriminating power of the set of terms. The upper cut-off threshold was used to exclude top ranked terms, that is terms that appear very

often in corpus documents. Such an example is the 2-word term *neural networks* that is contained in most of our documents. Moreover the bottom ranked terms are also excluded from the list of the useful terms since they are mostly noise created from the procedure of transforming the original postscript files to simple text files. This processing resulted in a list containing 2-, 3-, 4- and 5-word terms, which was then used to create the two different sets as follows.

Let $a_1 a_2 ... a_n$, where each a_i is a word, be a multi-word term from the aforementioned list, containing n words. A proximity formula is created out of this term in the following two ways:

1. $a_1 \prec_{[0,0]} a_2 \prec_{[0,0]} ... \prec_{[0,0]} a_n$. For each multi-word term in the list we introduce the proximity operator $\prec_{[0,0]}$ between the words of the multi-word term in order to create proximity formulas that represent strings. All the proximity formulas that are created this way form the first set of units named PF_0, which stands for *proximity formulas with word distance zero*. The number of operands in these proximity formulas varies according to the number of words contained in the multi-word terms. The minimum number of words in a multi-word term is obviously two, whereas the maximum is five. An example of a unit in this unit set is INVERSE $\prec_{[0,0]}$ DYNAMIC $\prec_{[0,0]}$ FUNCTION, which was produced from the term INVERSE DYNAMIC FUNCTION.

2. $a_1 \prec_{[0,k]} a_n$, where $1 \leqslant k \leqslant 10$. From each term in the list of multi-word terms we create proximity formulas with exactly two operands. These proximity formulas are created as follows. We replace *all* the middle words of the 3-, 4-, 5-word terms with the proximity operator $\prec_{[0,k]}$, specifying k to be a natural number drawn uniformly between 1 and 10. The choice of using a relatively small upper bound in the distance between two operands is inspired by the implementation of operator NEAR in well-known search engines such as AltaVista[3], where NEAR represents distance of ten words. All the proximity formulas created this way form the second set of units named PF_k, since they are proximity formulas with word distance k. An example of a unit in this set could be RBF $\prec_{[0,6]}$ NETWORKS, which could be created from the term e.g., RBF DYNAMIC DECAY ADJUSTMENT NETWORKS.

The second set of units used in the creation of our profiles database is the set of *nouns* that were extracted from document abstracts. The choice of nouns taken from document abstracts as opposed to the whole document can be justified by the argument that the abstracts are expected to be a brief description of the work carried out in the paper, thus very appropriate to describe the content of a paper. The procedure of creating the set of nouns is as follows. First we identified

[3]http://www.altavista.com

all the nouns in singular and plural form using the part-of-speech (POS) tagged abstracts that were available to us. After that we created a frequency-ranked list of these words and specified a upper and lower cut-off threshold to cut the most frequent and the least frequent words. The set of units that resulted from this procedure is denoted by NS, which stands for nouns.

The last set of units created is that of the authors' last names. We extracted all the names of the authors that were available to us from the corpus documents to obtain an author vocabulary V_{author} of 8833 last names. Using this author vocabulary to draw *uniformly* author names that will be used in a profile is not the best choice. This is because authors that are more active or produce more important papers than others are expected to be used heavier in profiles than the rest. The criterion for identifying the more important authors is how many citations they get from papers written by other researchers. In the citation graph corresponding to our corpus this is captured by the in-degree of the nodes for their paper as explained in [7]. The highest the in-degree for the papers of a specific author, the highest the probability for this author to appear in a profile. We define N_a to be the number of papers in the corpus that refer to at least one document of author a, and V_{author} the author vocabulary. N_a can easily be extracted from the full citation graph, and the author vocabulary is available to us from the NN corpus documents. The probability of author a to be used in a profile is given by:

$$P(a) = \frac{N_a}{\sum_{k \in V_{author}} N_k}$$

The above formula associates an author with the popularity of his writings, and thus with a probability degree of another researcher being interested in his work. As we can see, this set apart from containing the names of the authors that will be used for the profile creation, also contains information about the probability distribution that will be used for choosing the authors. This probability distribution is Zipf-like as it is shown in the log-log graph of Figure 5.5. This can be explained taking into account the general observation that in every scientific domain there exist a few heavily cited authors, while the rest receive less visibility (in terms of citations of their work). The unit set that is described above is denoted by AS, which stands for author surnames.

5.2.4 Details

Now that we have described the creation and the contents of the different unit sets, we are ready to give some details of how all this information is combined to create profiles. A profile under the subset of query language $\mathcal{AWP}$ supported by BF and PINDEX (see section 5.1) consists of a conjunction of atomic queries. These atomic queries can only be of the form $A \sqsupseteq wp$, where A is an attribute and wp is a conjunction of words and proximity formulas. In the rest of this section we

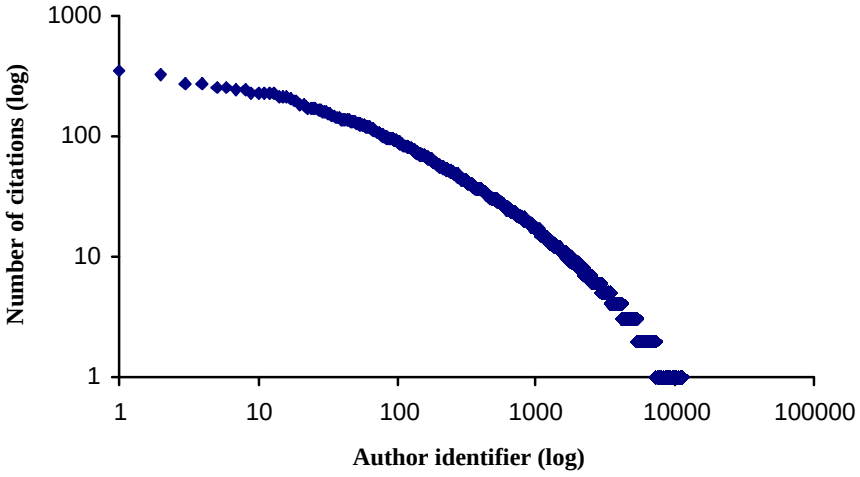

Figure 5.5: Distribution of citations among authors

will examine the different types of atomic queries that can be created according to the attributes that are available to us, that is AUTHOR, TITLE, ABSTRACT and BODY.

In our context creating a profile can be also looked at as the problem of choosing with a probability distribution between units contained in different sets. Not all the sets of units participate in the creation of an atomic formula of a specific attribute. Moreover different unit sets that participate in an atomic formula may have different *selection probabilities* in being chosen to participate in a profile. The unit sets that participate in the creation of an atomic formula, along with an indicatory value for the selection probability[4] of each unit are summarised in Table 5.4.

In general a creation of an atomic query is a 3-step process that can be described as follows:

1. Choose the number of units (or the *size* of an atomic query) S. This value is drawn uniformly from $[1, S_{max}]$, where S_{max} is the maximum number of units in an atomic query. S_{max} is defined to be two for atomic formulas of AUTHOR and TITLE attributes, whereas it is set to three for the ABSTRACT and BODY attributes. This differentiation in S_{max} is due to the different number of words contained typically in the different attributes of a document.

2. Taking into account the units that may participate in a specific atomic formula, we pick S units from these sets according to the selection probabilities

[4]Of course these values may vary depending on the wanted properties of the profile database to be generated.

Attribute:	Participating unit sets	Indicatory value of selection probability
TITLE	PF_0	0.4
	PF_k	0.4
	NS	0.2
ABSTRACT	PF_0	0.4
	PF_k	0.4
	NS	0.2
BODY	PF_0	0.4
	PF_k	0.4
	NS	0.2
AUTHOR	AS	1.0

Table 5.4: Participation of different unit sets in the creation of atomic formulas for each attribute

summarised in Table 5.4.

3. Having chosen these units, we take their conjunction to create the atomic formula.

For example an atomic formula for the TITLE attribute may be:

$$\text{TITLE} \sqsupseteq (RBF \prec_{[0,6]} NETWORKS) \wedge JAVA$$

which contains two units (remember that this is the maximum number of units allowed for the TITLE attribute): unit (RBF $\prec_{[0,6]}$ NETWORKS) drawn from unit set PF_k and unit JAVA drawn from unit set NS. Changing the selection probabilities of different unit sets results in changing how often units of a specific set will appear in atomic queries of the corresponding attribute. Other possible atomic formulas could be:

$$\text{TITLE} \sqsupseteq IMPLEMENTATION \wedge (INVERSE \prec_{[0,0]} DYNAMIC \prec_{[0,0]} FUNCTION)$$

$$\text{TITLE} \sqsupseteq REAL \prec_{[0,0]} WORLD \prec_{[0,0]} APPLICATION$$

$$\text{TITLE} \sqsupseteq ALGORITHM \wedge IMPLEMENTATION$$

Atomic queries for ABSTRACT and BODY attributes are created in a similar way. The only differentiations between atomic formulas for different attributes are different selection probabilities for the unit sets and different maximum atomic query sizes.

At the same time creating atomic queries for attribute AUTHOR is somewhat different. Atomic queries for attribute AUTHOR can have a maximum size of two units and can contain either one unit or a conjunction of two units from unit set

AS. Note that for the case of an atomic query for the AUTHOR attribute using more words in conjunction would make the profile very specific, thus not suitable for an information alert setting. Note also that proximity operations could also be used in these atomic queries (e.g., JOHN $\prec_{[0,0]}$ BROWN). However the authors first names were not available from the corpus documents so this option was not adopted. Below are some examples of such atomic queries:

$$\text{AUTHOR} \sqsupseteq BROWN$$

$$\text{AUTHOR} \sqsupseteq SMITH \wedge JOHNSON$$

As we have already mentioned, in the subset of the language that is supported by the two implemented algorithms, a profile consists of a conjunction of such atomic queries. What is needed now is a way to select which atomic queries will be introduced in each profile. To do so we assign selection probabilities to each one of the four types of atomic queries and according to this selection probabilities we decide which atomic query be included in a single profile and which will not. Each type of atomic atomic queries is (or is not) included in a profile independently of the rest of the types. For example for a specific profile generation scenario if the selection probability of all four types of atomic queries is 85% then atomic queries for the AUTHOR attribute will appear in the 85% of the profiles in the profile database. The same holds for the rest of the attribute types (TITLE, ABSTRACT and BODY). At this point we should stress that in this way all possible combinations of atomic queries can appear in generated profiles, and that a simple probability calculation allows us to control or even exclude[5] certain types of atomic queries from the generated profiles.

5.3 Experimental Evaluation

Having specified how the profile database is created we now move to the experimental evaluation of our main memory algorithms. In this section we present four different experiments that clearly demonstrate the advantages and disadvantages of each algorithm.

5.3.1 Settings

The algorithms described in section 5.1 were implemented in C, and all the experiments were run on a PC, with a Pentium III 1.7GHz processor, with 1GB RAM, running Linux. The time shown in the graphs is elapsed time in milliseconds and no other processes were run on the PC during the experiments. For each of our conducted experiments we followed a standard procedure described below.

[5]By setting the selectivity probability of this type to a very low number or even zero.

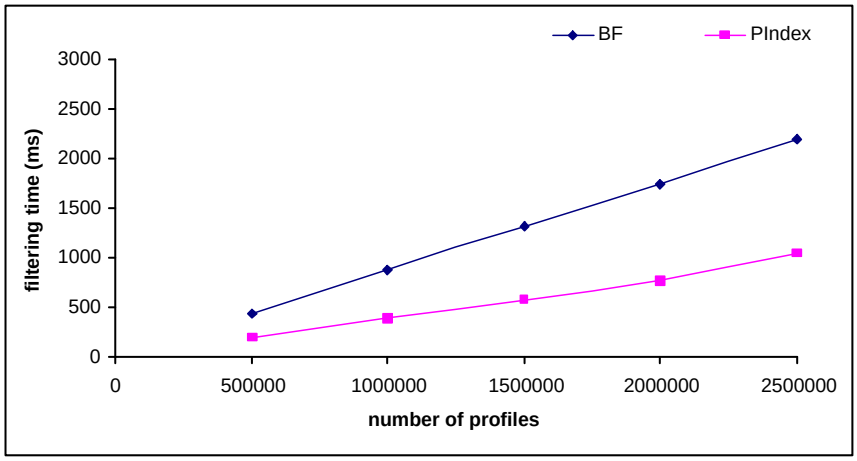

Figure 5.6: Effect of the profiles database size in filtering time

Initially we generate the database of profiles and select the documents that will be used as an input for the algorithms. Both the profile creation and the document selection depend on the nature of the experiment and the targeting behaviour of the algorithms. Having decided on these, we populate the data structures of each algorithms, load the document into main memory and run each algorithm. All the experiments are run five times and the results are averaged to eliminate any fluctuations in the time measurements. Note that when we refer to the matching time of a document against a database of profiles we mean only the time needed by the algorithms to discover which profiles are satisfied by the incoming document. This time does not include the time needed to upload the document from the secondary storage to the main memory, or the time to construct the document index. Finally we have to note that an important factor affecting the matching time for a profile is how many atomic formulas it contains. The profiles we used for the experiments described in this chapter had an average of 3.5 atomic formulas per profile (remember that our attribute universe for these experiments consists of four attributes). This means that we used "heavily loaded" profiles something that also affected the mean matching time (this is demonstrated clearly in Section 5.3.4).

5.3.2 Varying the Number of Profiles

The first experiment that we conducted to evaluate our algorithms, targeted the performance of the two algorithms under different sizes of profiles databases. In this experiment we randomly selected one hundred document from the NN corpus and used them as input document in the profile databases of different sizes (but of the same general properties). The size and the matching percentage for each document used was different but the average document size was 6869

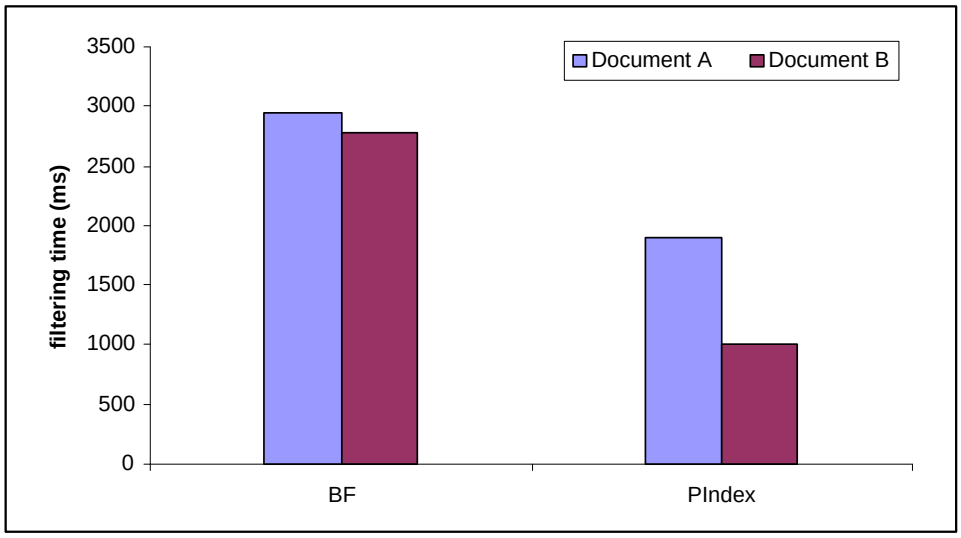

Figure 5.7: Effect of the profiles' matching percentage in filtering time

words whereas the average matching percentage for the one hundred documents was around 1%.

As we can see in Figure 5.6 both algorithms are linear to the size of the profiles database, however PIndex seems to be less sensitive in the change of the profile database size. Moreover due to the index structures used by PIndex we can observe a significant improvement in the matching time compared to the sequential scan method BF. PIndex seems to speedup the matching procedure by a factor of 2, managing to filter as much as 2.5 million profiles in less than one second.

5.3.3 Varying the Matching Percentage

In this experiment we wanted to observe the behaviour of the two algorithms when the number of profiles that match an incoming document increases. That is we wanted to examine how sensitive each algorithm is to the matching degree of profiles. For this experiment we used two document with about the same size and the same attributes. However the matching percentage of the document was different for the same 2.5 million profiles. More specifically we conducted the experiment comparing the behaviour of the algorithms in two documents. Document A was 7055 words in size and had a matching percentage of about 1%, whereas document B contained 7059 words and had a matching percentage of 0.4%. Both documents contained four $(attribute, value)$ pairs (remember that the size of the attribute universe is four), and there were no significant variations in the size (in words) of the corresponding values between the two documents.

Many interesting conclusions can be drawn about the performance of the two

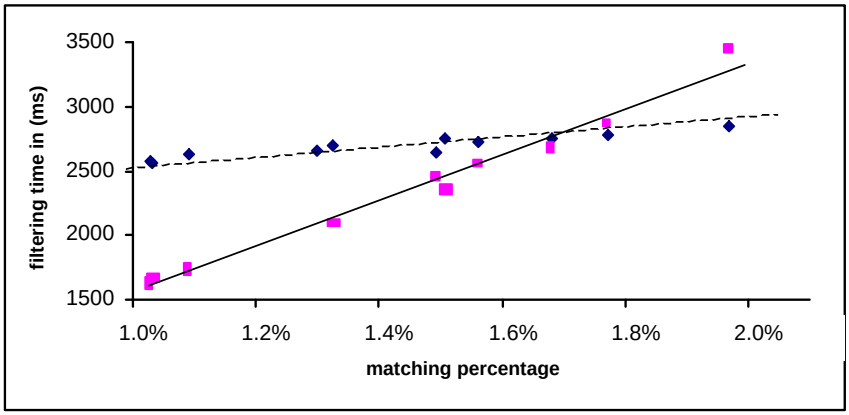

Figure 5.8: Sensitivity of each algorithm to matching percentage of profiles

algorithms under this evaluation scenario. As it is shown in figure 5.7, BF is relatively insensitive to the change in the matching percentage, as opposed to PIndex which shows remarkable improvement at lower matching percentage. More specifically the reduction by 150% of the matching percentage results in a reduction of only 5% for BF, whereas the corresponding reduction for PIndex is nearly 50%. This can be explained by the fact that BF scans sequentially all the profiles to discover those that are satisfied by the incoming document. This time is relatively big compared to the overhead added to the algorithm to perform some extra test for the case of more matching profiles. On the other hand, PIndex achieves low filtering times for small matching percentages, and the overhead imposed by the extra matching tests has a more visible effect. This overhead is mainly due to the increase in the similarity between the profiles. This similarity results in the searching of longer lists at each hash table slot thus adding extra cost to match the indexed profiles.

The sensitivity of PIndex can also be seen in Figure 5.8 where we can see that the two algorithms behave very differently. BF shows a remarkable stability to matching percentage changes, whereas PIndex seems to degrade in performance as the matching percentage increases.

This behaviour of PIndex shows one of the main weaknesses of the specific indexing scheme. Imagine a scenario where PIndex is used in a news alert system and a sudden crisis occurs (e.g., an earthquake or terrorist act). This would result in users posting many similar profiles thus causing a serious degrade in the performance of the algorithm.

5.3.4 Varying the Average Number of Atomic Formulas

The structure and the size of the profiles themselves are expected to have a significant effect on the filtering time for both algorithms. To verify this observation

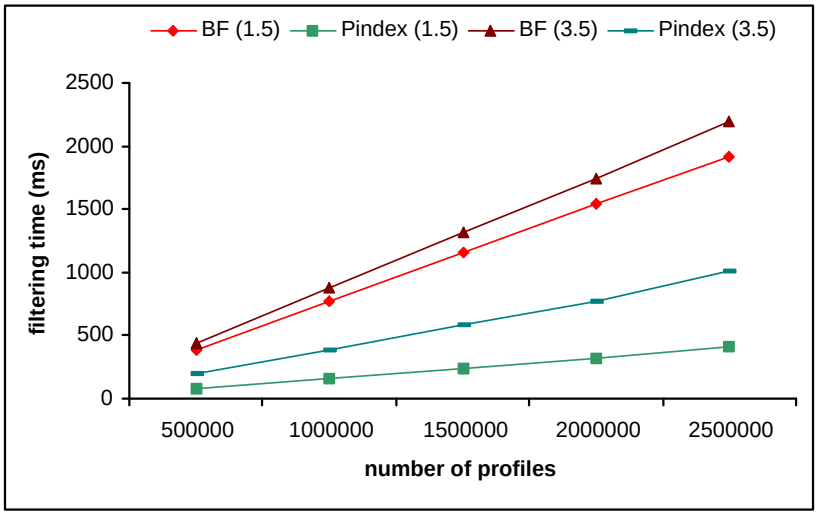

Figure 5.9: Effect of the number of atomic formulas in a profile

and also to see how the nature of the stored profiles affects the performance of each algorithm, we conducted an experiment were we use a profile database with "simpler" profiles. This means that we defined the profile generation parameters, so as to reduce the selection probabilities for atomic formulas (see Section 5.2.4) to create profiles with less atomic formulas per profile. In this experiment an average of 1.5 atomic formulas per profile is used and the filtering times for both algorithms are compared with those presented in Figure 5.6 where the average is 3.5. In Figure 5.9 we can see both this experiment and the results from the experiment in Section 5.3.2.

As we can easily observe our conjecture about the difference in the filtering time between profile databases with different characteristics is confirmed. Both algorithms seem to be influenced by the different nature of the profiles, something that was expected since they have an "easier" task with the profile set that contains fewer atomic formulas per profile. However as we can see BF is much less affected from this change in the profile characteristics than PINDEX. This is because the sequential scan of the profiles is a time consuming operation and consumes most of the filtering time, thus not leaving other parameters (such as the number of atomic formulas in a profile) to play an important role. On the other hand we can see a significant improvement on the performance of PINDEX. This can be explained as follows. Specifying the profiles to have three or four atomic formulas results in a considerable overload of the index structure in terms of the size of lists in each hash table slot. This overload of the index structures results in a decrease in the performance of the algorithm. On the other hand specifying profiles to have less atomic formulas (e.g., one of two) results less load on the index structure (since the atomic formulad are dispersed over the different

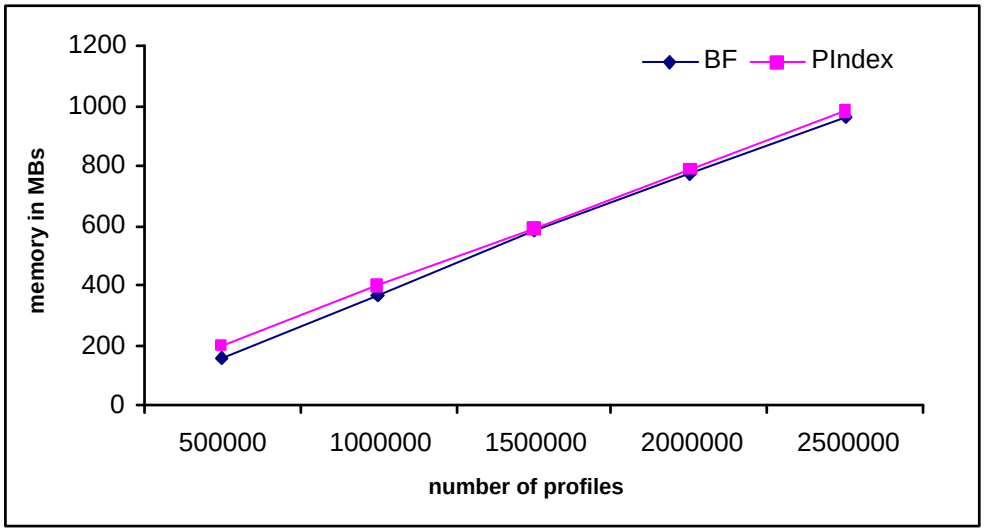

Figure 5.10: Memory usage for different sizes of the profile database

WDs of PINDEX). Of course this behaviour of PINDEX can be also seen as a weakness of the algorithm. Ideally we would like the performance of the index structures to be efficient and yet insensitive to changes in the nature of the data indexed.

5.3.5 Memory Usage

The main objective of this experiment was to see what is the additional space cost that we have to pay due to the index structures used by PINDEX. This experiment was carried out with several size varying profile databases with the same general characteristics. The evaluation was carried out using a relatively big document (about 27000 words) that would be considerably demanding in terms of its index structures.

As we can see in Figure 5.10 both BF and PINDEX are linear in terms of the memory usage. Moreover we can observe that the index structures utilised by PINDEX do not impose a great space cost and both algorithms seem to need about the same amount of memory for performing a filtering task. This happens because apart from the two tables (TOTAL and COUNT) that are used for the final comparison of how many atomic formulas of a profile match, no other extra information is kept for each profile. Finally the index that is built upon the incoming documents is apparently not space consuming even for large documents. This is because we are indexing only the distinct words that exist in an incoming document, thus gaining a significant reduction in the storage cost.

5.4 Summary

In this chapter we have presented in detail two main memory algorithms that support profile filtering under the model $\mathcal{AWP}$, presented in Section 3.4. We have also evaluated these algorithms experimentally using real documents and realistic user profiles.

Chapter 6

Concluding Remarks

In this work we dealt with the problem of textual information dissemination in the context of distributed P2P systems. We put our main focus on data models and query languages especially designed for information dissemination, when information is in the form of text. We also identified problems that frequently arise in such a scenario and studied their computational complexity. Finally we designed, implemented and experimentally evaluated two algorithms for the problem of filtering as it arises in textual information dissemination.

Initially our study surveyed the area of agent middleware and paid specific attention to information management applications. To better position ourselves regarding other areas of related research, we also discussed in detail some popular Internet systems. We showed that these systems can be considered to be multi-agent systems with a central role for the middleware.

Then we focused on defining data models and query languages to express documents and user profiles. We incrementally developed three such models and query languages moving from very simple ones to ones with more expressive power. Our work in this field extends traditional concepts that are already known in information retrieval [10, 27, 28], and complements recent proposals for querying textual information dissemination in event-based systems [20, 19]. It is our belief that such models and languages can be of great use to applications like alert systems for digital libraries or news dissemination systems.

Having defined the languages for information dissemination, we then identified four fundamental problems of all information dissemination systems (satisfiability, satisfaction, filtering and entailment) and studied their computational complexity.

Finally, we proposed two main memory algorithms that solve the problem of filtering in information dissemination under the data model $\mathcal{AWP}$ [72, 73]. The first algorithm adopts a brute force strategy and was mainly implemented for comparison purposes. The second algorithm adopts a two dimensional indexing on user profiles. To prove our case we evaluated both algorithms under a realistic scenario using documents from the NN corpus [74, 40] and realistic profiles created

from terms extracted from the documents.

6.1 Future Work

There are many problems that need to be addressed in future work. We are already in the process of designing new algorithms for the problem of filtering that take into consideration *commonalities* among user profiles. We are also currently implementing a new algorithm for the case of the model $\mathcal{AWPS}$.

On the language front, we would like to extend our data models and languages to cope also with other types of information, for example prices. This can be useful in many applications (e.g., e-commerce) and can easily be done by introducing attributes of numeric type and comparison operators such as "less than" [19, 20]. Moreover we envision moving towards a document representation based on XML and a profile language based on XPath [136] or XQuery [137]. Designing efficient filtering algorithms for such languages is a difficult task that could be tackled using ideas from recent work such as [6, 24]. Finally, architecting peer-to-peer systems that allow for efficient retrieval and dissemination of information is a difficult problem and a lot of research remains to be done in this area. For recent work by our research group in this area see [133].

Bibliography

[1] ResearchIndex: The NEC Research Institute scientific literature digital library. http://www.researchindex.com.

[2] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web - From Relations to Semistructured Data and XML*. Morgan Kaufmann Publishers, 2000.

[3] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison Wesley, 1995.

[4] M.K. Aguilera, R.E. Strom, D.C. Sturman, M. Astley, and T.D. Chandra. Matching Events in a Content-based Subscription System. In *Proceedings of the Eighteenth Annual ACM Symposium on Principles of Distributed Computing (PODC '99)*, pages 53–62, New York, May 1999. Association for Computing Machinery.

[5] A. Aho, R. Sethi, and J. Ullman. *Compilers, principles, techniques, and tools*. Addison Wesley, 1986.

[6] M. Altinel and M.J. Franklin. Efficient Filtering of XML Documents for Selective Dissemination of Information. In Amr El Abbadi, Michael L. Brodie, Sharma Chakravarthy, Umeshwar Dayal, Nabil Kamel, Gunter Schlageter, and Kyu-Young Whang, editors, *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10–14, 2000, Cairo, Egypt*, pages 53–64, Los Altos, CA 94022, USA, 2000. Morgan Kaufmann Publishers.

[7] Y. An, J. Janssen, and E. Milios. Characterizing and Mining the Citation Graph of the Computer Science Literature. Technical Report CS-2001-02, Dalhousie University, Canada, 26 September 2001.

[8] V. Arens, C. Y. Chee, C. N. Hsu, and C. A. Knoblock. Retrieving and Integrating Data from Multiple Information Sources. *International Journal on Intelligent and Cooperative Information Systems*, 2(2):127–158, 1993.

[9] J.L. Austin. *How to Do Things With Words*. Harvard University Press, Cambridge, MA, 1962.

[10] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, 1999.

[11] S. Baker. *CORBA Distributed Objects Using Orbix*. Addison-Wesley and ACM Press, 1997.

[12] R. J. Bayardo, W. Bohrer, R. Brice, A. Cichocki, J. Fowler, A. Helal, V. Kashyap, T. Ksiezyk, G. Martin, M. Nodine, M. Rashid, M. Rusinkiewicz, R. Shea, C. Unnikrishnan, A. Unruh, and D. Woelk. InfoSleuth: agent-based semantic integration of information in open and dynamic environments. In Joan M. Peckman, editor, *Proceedings, ACM SIGMOD International Conference on Management of Data: SIGMOD 1997: May 13–15, 1997, Tucson, Arizona, USA*, volume 26(2) of *SIGMOD Record (ACM Special Interest Group on Management of Data)*, pages 195–206, New York, NY 10036, USA, 1997. ACM Press.

[13] S. Bergamaschi. Extraction of Information from Highly Heterogeneous Source of Textual Data. In Peter Kandzia and Matthias Klusch, editors, *Proceedings ot the First International Workshop on Cooperative Information Agents*, volume 1202 of *LNAI*, pages 42–63, Berlin, February 26–28 1997. Springer.

[14] C. Bowman, P. B. Danzig, D.R. Hardy, U. Manber, and M. F. Schwartz. Scalable internet resource discovery: research problems and approaches. *Comm. A.C.M.*, 37(8):98–107, August 1994.

[15] C.M. Bowman, P.B. Danzig, D.R. Hardy, Udi Manbe, Michael F. Schwartz, and Duane P. Wessels. Harvest: A Scalable, Customizable Discovery and Access System. Technical Report CU-CS-732-94, Department of Computer Science, University of Colorado, Boulder, August 1994.

[16] Eric Brill. A simple rule-based part-of-speech tagger. In *Proceedings of ANLP-92, 3rd Conference on Applied Natural Language Processing*, pages 152–155, Trento, 1992. URL: citeseer.nj.nec.com/brill92simple.html.

[17] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, April 1998.

[18] James P. Callan, W. Bruce Croft, and Stephen M. Harding. The INQUERY retrieval system. In *Proceedings of the International Conference Database and Expert Systems Applications (DEXA'92)*, pages 78–83, 1992.

[19] A. Campailla, S. Chaki, E. Clarke, S. Jha, and H. Veith. Efficent filtering in publish-subscribe systems using binary decision diagrams. In *Proceedings*

of the 23rd International Conference on Software Engineering, Toronto, Ontario, Canada, 2001.

[20] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Achieving scalability and expressiveness in an internet-scale event notification service. In *Proceedings of the 19th ACM Symposium on Principles of Distributed Computing (PODC'2000)*, pages 219–227, 2000.

[21] A. Carzaniga, D.S. Rosenblum, and A.L. Wolf. Achieving Scalability and Expressiveness in an Internet-scale Event Notification Service. In *Proceedings of the 19th Annual ACM Symposium on Principles of Distributed Computing (PODC-00)*, pages 219–228, NY, July 16–19 2000. ACM Press.

[22] A. Cassandra, D. Chandrasekara, and M. Nodine. Capability-Based Agent Matchmaking. In Carles Sierra, Maria Gini, and Jeffrey S. Rosenschein, editors, *Proceedings of the Fourth International Conference on Autonomous Agents*, pages 201–202, Barcelona, Catalonia, Spain, June 2000. ACM Press. Poster announcement.

[23] U. Çetintemel, M. Franklin, and C. Giles. Self-Adaptive User Profiles for Large-Scale Data Delivery. In *17th International Conference on Data Engineering (ICDE' 00)*, Heidelberg - Germany, May 2000. IEEE.

[24] C.-Y. Chan, W. Fan, P. Felber, M. Garofalakis, and R. Rastogi. Tree Pattern Aggregation for Scalable XML Data Dissemination. In *Proceedings of the 28th VLDB Conference, Hong Kong, China*, 2002.

[25] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing*, pages 77–90, 1977.

[26] C.-C. K. Chang. *Query and Data Mapping Across Heterogeneous Information Sources*. PhD thesis, Stanford University, January 2001.

[27] C.-C. K. Chang, H. Garcia-Molina, and A. Paepcke. Boolean Query Mapping across Heterogeneous Information Sources. *IEEE Transactions on Knowledge and Data Engineering*, 8(4):515–521, 1996.

[28] C.-C. K. Chang, H. Garcia-Molina, and A. Paepcke. Predicate Rewriting for Translating Boolean Queries in a Heterogeneous Information System. *ACM Transactions on Information Systems*, 17(1):1–39, 1999.

[29] D. Chauhan and A.D. Baker. JAFMAS: A Multiagent Application Development System. In Katia P. Sycara and Michael Wooldridge, editors, *Proceedings of the 2nd International Conference on Autonomous Agents (Agents'98)*, pages 100–107, New York, May 9–13, 1998. ACM Press.

[30] T. T. Chinenyanga and N. Kushmerick. Expressive retrieval from XML documents. In *Proceedings of SIGIR'01*, September 2001.

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

[32] Ian Clarke, O. Sandberg, B. Wiley, and T.W. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. In *Workshop on Design Issues in Anonymity and Unobservability*, July 2000.

[33] P.R. Cohen and H.J. Levesque. Communicative Actions for Artificial Agents. In Jeffrey M. Bradshaw, editor, *Software Agents*, chapter 18, pages 419–436. AAAI Press / The MIT Press, 1997.

[34] William W. Cohen. WHIRL: A word-based information representation language. *Artificial Intelligence*, 118(1-2):163–196, 2000.

[35] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

[36] R. S. Cost, T. Finin, Y. Labrou, X. Luan, Y. Peng, I. Soboroff, J. Mayfield, and A. Boughanam. Jackal: A Java-based Tool for Agent Development. In *AAAI-98, Workshop on Tools for Agent Development*, Madison, WI, July 1998.

[37] R. Davis and R. Smith. Negotiation as a Metaphor for Distributed Problem Solving. *Artificial Intelligence*, 20(1):63–109, January 1983.

[38] K. Decker and K. Sycara. Intelligent adaptive information agents. *Intelligent Information Systems*, 9(3), 1997.

[39] K. Decker, K. Sycara, and M. Williamson. Middle-agents for the internet. In *Proceedings of IJCAI-97*, 1997.

[40] L. Dong. Automatic term extraction and similarity assessment in a domain specific document corpus. Master's thesis, Department of Computer Science, Dalhousie University, Halifax, Canada, 2002.

[41] M. Koubarakis et. al. Project DIET Deliverable 7 (Information Brokering), December 2001.

[42] F. Fabret, H. A. Jacobsen, F. Llirbat, J. Pereira, K. A. Ross, and D. Shasha. Filtering algorithms and implementation for very fast publish/subscribe systems. In *Proceedings of ACM SIGMOD-2001*, 2001.

[43] F. Fabret, F. LLirbat, J. Pereira, and D. Shasha. Publish/Subscribe on the Web at Extreme Speed. In *Proceedings of ACM SIGMOD Conf. on Management of Data*, 2000.

[44] D. Faensen, L. Faulstich, H. Schweppe, A. Hinze, and A. Steidinger. Hermes – A Notification Service for Digital Libraries. In *Proceedings of the Joint ACM/IEEE Conference on Digital Libraries (JCDL'01), Roanoke, Virginia, USA*, 2001.

[45] D. Fensel. The semantic Web and its languages. *Trends and Controversies Column. IEEE Intelligent Systems*, 15(6), November 2000.

[46] T. Finin, R. Fritzson, D. McKay, and R. McEntire. KQML as an Agent Communication Language. In N. Adam, B. Bhargava, and Y. Yesha, editors, *Proceedings of the 3rd International Conference on Information and Knowledge Management (CIKM'94)*, pages 456–463, Gaithersburg, MD, USA, 1994. ACM Press.

[47] T. Finin, Y. Labrou, and J. Mayfield. KQML as an agent communication language. In Jeffrey M. Bradshaw, editor, *Software Agents*, chapter 14, pages 291–316. AAAI Press / The MIT Press, 1997.

[48] T. Finin, J. Weber, G. Wiederhold, M. Genesereth, R. Fritzson, D. McKay, J. McGuire, R. Pelavin, S. Shapiro, and C. Beck. Draft specification of the KQML Agent-Communication Language, 1993.

[49] T.W. Finin and Y. Labrou. Napster as a Multi-Agent System. Presentation at the 18th FIPA meeting, University of Maryland Baltimore County, July 2000.

[50] P.W. Foltz and S.T. Dumais. Personalised information delivery: An analysis of information filtering methods. *Communications of the ACM*, 35(12):29–38, 1992.

[51] M. J. Franklin and S. B. Zdonik. "Data In Your Face": Push Technology in Perspective. In *Proceedings ACM SIGMOD International Conference on Management of Data*, pages 516–519, 1998.

[52] K. Frantzi, S. Ananiadou, and H. Mima. Automatic recognition of multi-word terms:the C-value/NC-value method. *International Journal on Digital Libraries*, 5(2), 2000.

[53] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. D. Ullman, V. Vassalos, and J. Widom. The TSIMMIS Approach to Mediation: Data Models and Languages. *Journal of Intelligent Information Systems*, 8(2):117–132, 1997.

[54] M. R. Genesereth and S. P. Ketchpel. Software agents. *Communications of the ACM*, 37(4), July 1994.

[55] Michael R. Genesereth. An agent-based framework for interoperability. In Jeffrey M. Bradshaw, editor, *Software Agents*, chapter 15, pages 317–346. AAAI Press / The MIT Press, 1997.

[56] M.R. Genesereth, A.M. Keller, and O.M. Duschka. Infomaster: an information integration system. In Joan M. Peckman, editor, *Proceedings, ACM SIGMOD International Conference on Management of Data: SIGMOD 1997: May 13–15, 1997, Tucson, Arizona, USA*, volume 26(2) of *SIGMOD Record (ACM Special Interest Group on Management of Data)*, pages 539–542, New York, NY 10036, USA, 1997. ACM Press.

[57] J.A. Giampapa, M.Paolucci, and K. Sycara. Agent Interoperation across Multiagent System Boundaries. In Carles Sierra, Maria Gini, and Jeffrey S. Rosenschein, editors, *Proceedings of the Fourth International Conference on Autonomous Agents*, pages 179–186, Barcelona, Catalonia, Spain, June 2000. ACM Press.

[58] J. Goguen, D. Nguyen, J. Meseguer, Luqi, D. Zhang, and V. Berzins. Software Component Search. *Journal of Systems Integration*, 6:93–134, 1996.

[59] B. Grosof and Y. Labrou. An approach to using XML and a rule-based content language with an Agent Communication Language. In *Workshop on Agent Communication Languages, IJCAI-99*, Stockholm, Sweden, July 1999.

[60] T. Gruber. A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, 6(4):199–121, 1993.

[61] H. Gupta and D. Srivastava. The data warehouse of newsgroups. In *Proceedings of the 7th International Conference on Database Theory (ICDT '99)*, volume 1540 of *Lecture Notes in Computer Science*, pages 471–488. Springer, 1999.

[62] D. Heimbigner. Adapting Publish/Subscribe Middleware to Achieve Gnutella-like Functionality. In *ACM Symposium on Applied Computing (SAC 2001): Special Track on Coordination Models, Languages and Applications*, Las Vegas, NV, March 11–14 2001.

[63] J. Hendler and D. McGuinness. The DARPA Agent Markup Language. *Trends and Controversies Column. IEEE Intelligent Systems*, 15(6), November 2000.

[64] H. V. Jagadish, L. V. S. Lakshmanan, T. Milo, D. Srivastava, and D. Vista. Querying network directories. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 133–144, 1999.

[65] S. Jha, P. Chalasani, O. Shehory, and K. Sycara. A Formal Treatment of Distributed Matchmaking. In Katia P. Sycara and Michael Wooldridge, editors, *Proceedings of the 2nd International Conference on Autonomous Agents (Agents'98)*, pages 457–458, New York, May 9–13, 1998. ACM Press.

[66] A.M. Julienne and B. Holtz. *ToolTalk and open protocols, inter-application communication.* Englewood Cliffs, NJ, 1994.

[67] Anthony Klug. On Conjunctive Queries Containing Inequalities. *Journal of ACM*, 35(1):146–160, 1988.

[68] C. A. Knoblock and S. Minton. The Ariadne approach to web-based information integration. *IEEE Intelligent Systems*, 13(5), September 1998.

[69] M. Koubarakis. Boolean Queries with Proximity Operators for Information Dissemination. Proceedings of the Workshop on Foundations of Models and Languages for Information Integration (FMII-2001), Viterbo, Italy , 16-18 September, 2001. In LNCS (forthcoming).

[70] M. Koubarakis. Textual Information Dissemination in Distributed Event-Based Systems. Proceedings of the International Workshop on Distributed Event-Based systems (DEBS'02), July 2-3, 2002, Vienna, Austria.

[71] M. Koubarakis, T. Koutris, P. Raftopoulou, and C. Tryfonopoulos. Efficient Agent-Based Dissemination of Textual Information. In *2nd Hellenic Conference on Artificial Intelligence (SETN 02)*, Thessaloniki, Greece, 11-12 April 2002.

[72] M. Koubarakis, T. Koutris, C. Tryfonopoulos, and P. Raftopoulou. Information Alert in Distributed Digital Libraries: The Models, Languages and Architecture of DIAS. In *Proceedings of the 6th European Conference on Digital Libraries (ECDL2002)*, September 2002.

[73] M. Koubarakis, C. Tryfonopoulos, P. Raftopoulou, and T. Koutris. Data models and languages for agent-based textual information dissemination. In *Proceedings of the 6th International Workshop on Cooperative Information Agents(CIA2002), Madrid*, Lecture Notes in Computer Science. Springer, 2002.

[74] T. Koutris. Textual Information Dissemination in Distributed Agent Systems: Architectures and Efficient Filtering Algorithms. Master's thesis, Department of Electronic and Computer Engineering, Technical University of Crete, Greece. (forthcoming) .

[75] S. Kumar and P.R. Cohen. Towards a fault-tolerant multi-agent system architecture. In Carles Sierra, Gini Maria, and Jeffrey S. Rosenschein, editors, *Proceedings of the 4th International Conference on Autonomous Agents (AGENTS-00)*, pages 459–466, NY, June 3–7 2000. ACM Press.

[76] S. Kumar, P.R. Cohen, and H.J. Levesque. The Adaptive Agent Architecture: Achieving Fault-Tolerance Using Persistent Broker Teams. In *Proceedings of the 4th International Conference on Multi-Agent Systems*, pages 159–166, July 7–12 2000.

[77] D. Kuokka and L. Harada. A communication infrastructure for concurrent engineering. *Artificial Intelligence in Engineering, Design, Analysis and Manufacturing*, 1995.

[78] D. Kuokka and L. Harada. Matchmaking for information agents. In Chris S. Mellish, editor, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 672–679, Montreal, Quebec, Canada, August 1995. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA.

[79] D. Kuokka and L. Harada. Issues and Extensions for Information Matchmaking Protocols. *International Journal of Cooperative Information Systems*, 5(2-3):251–273, 1996.

[80] D. R. Kuokka and L. P. Harada. Issues and extensions for information matchmaking protocols. *International Journal of Cooperative Information Systems*, 5(2-3):251–274, 1996.

[81] Y. Labrou. *Semantics for an Agent Communication Language*. PhD thesis, Department of Computer Science and Electrical Engineering, University of Maryland, Baltimore County, 1996.

[82] Y. Labrou and T. Finin. Semantics and Conversations for an Agent Communication Language. In Martha E. Pollack, editor, *Proceedings of the Fifteenth International Joint Conference on Artifical Intelligence (IJCAI-97)*, pages 584–591. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA, 1997.

[83] Y. Labrou, T. Finin, and Y. Peng. Agent Communication Languages: The Current Landscape. *IEEE Intelligent Systems*, 14(2):45–52, March/April 1999.

[84] Yannis Labrou and Tim Finin. A semantics approach for KQML-a general purpose communication language for software agents. In *3rd International Conference on Information and Knowledge Management*, November 1994.

[85] Steve Lawrence, C. Lee Giles, and Kurt Bollacker. Digital libraries and autonomous citation indexing. *IEEE Computer*, 32(6):67–71, 1999.

[86] E. D. Lazowska, J. L. Zahorjan, G. S. Graham, and K. C. Sevcik. *Quantitative System Performance*. Prentice-Hall, 1984.

[87] A.Y. Levy, D. Srivastava, and T. Kirk. Data Model and Query Evaluation in Global Information Systems. *Journal of Intelligent Information Systems - Special Issue on Networked Information Discovery and Retrieval*, 5(2):121–143, 1995.

[88] S.H. Li and P.B. Danzig. Boolean similarity measures for resource discovery. *IEEE Transactions on Knowledge and Data Engineering*, 9(6):863–876, November/December 1997.

[89] Shih-Hao Li and Peter B. Danzig. Boolean similarity measures for resource discovery. *TKDE*, 9(6):863–876, 1997.

[90] J. Lu, J. Mylopoulos, and J. Ho. Towards Extensible Information Brokers Based on XML. In *Proceedings of CAISE2000*, pages 32–46, Stockholm, Sweden, June 5–9 2000.

[91] C.D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. The MIT Press, Cambridge, Massachusetts, 1999.

[92] D. Martin, A. Cheyer, and D. Moran. The Open Agent Architecture: a framework for building distributed software systems. *Applied Artificial Intelligence*, 13(1-2):91–128, 1999.

[93] P. C. Mitchell. A note about the proximity operators in information retieval. In *Proceedings of SIGIR'73*, pages 177–179, 1973.

[94] T. Mohri and Y. Takada. Virtual Integration of Distributed Database by Multiple Agents. In Setsuo Arikawa and Hiroshi Motoda, editors, *Proceedings of the 1st International Conference on Discovery Science (DS-98)*, volume 1532 of *LNAI*, pages 413–414, Berlin, December 14–16 1998. Springer.

[95] E. Moss, editor. *Data Engineering – Special Issue on Integrating Text Retrieval and Databases*, volume 19, March 1996.

[96] A. Moukas. Amalthaea: Information Discovery and Filtering using a Multiagent Evolving Ecosystem. *Applied Artificial Intelligence: An International Journal*, 11(5):437–457, 1997.

[97] A. Moukas and G. Zacharia. Evolving a Multiagent Information Filtering Solution in Amalthaea. In W. Lewis Johnson and Barbara Hayes-Roth, editors, *Proceedings of the First International Conference on Autonomous*

Agents (Agents'97), pages 394–403, New York, February 5–8, 1997. ACM Press.

[98] G. Navarro and R. A. Baeza-Yates. Proximal Nodes: A Model to Query Document Databases by Content and Structure. *ACM Transactions on Information Systems*, 15(4):400–435, 1997.

[99] J. Nunez-Suarez, D. O'Sullivan, H. Brouchoud, P. Cros, C. Moore, and C. Byrne. Experiences in the use of FIPA agent technologies for the development of a personal travel application. In *Proceedings of Agents 2000*, pages 357–364, Barcelona, Spain, 2000.

[100] E. Ogston and S. Vassiliadis. Matchmaking among minimal agents without a facilitator. In Jörg P. Müller, Elisabeth Andre, Sandip Sen, and Claude Frasson, editors, *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 608–615, Montreal, Canada, May 2001. ACM Press.

[101] M. Paolucci, Z. Niu, K. Sycara, C. Domashnev, S. Owens, and M. Van Velsen. Matchmaking to Support Intelligent Agents for Portfolio Management. In *Proceedings of the 7th Conference on Artificial Intelligence (AAAI-99) and of the 12th Conference on Innovative Applications of Artificial Intelligence (IAAI-99)*, pages 1125–1126, Menlo Park, CA, July 30–3 1999. AAAI Press.

[102] C. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[103] J. Pereira, F. Fabret, F. Llirbat, and D. Shasha. Efficient matching for web-based publish/subscribe systems. In *Proceedings of COOPIS-2000*, 2000.

[104] U. Pfeifer, N. Fuhr, and T. Huynh. Searching Structured Documents with the Enhanced Retrieval Functionality of freeWAIS-sf and SFgate. In *Proceedings of the 3rd International World-Wide Web Conference*, pages 1027–1036, 1995.

[105] J. Pitt, F. Guerin, and C. Stergiou. Protocols and Intentional Specifications of Multi-Party Agent Conversations for Brokerage and Auctions. In Carles Sierra, Maria Gini, and Jeffrey S. Rosenschein, editors, *Proceedings of the Fourth International Conference on Autonomous Agents*, pages 269–276, Barcelona, Catalonia, Spain, June 2000. ACM Press.

[106] J. Pitt and A. Mamdani. A Protocol-Based Semantics for an Agent Communication Language. In *Proceedings of International Joint Conference on Artificial Intelligence*, pages 486–491, Stocholm, Sweden, 1999.

[107] J. Pitt and A. Mamdani. Some Remarks on the Semantics of FIPA's Agent Communication Language. *Autonomous Agents and Multi-Agent Systems*, 2(4):486–491, November 1999.

[108] M.F. Porter. An Algorithm for Suffix Striping. *Program*, 14(3):130–137, 1980.

[109] A.S. Rao and M.P. Georgeff. Modeling rational agents within a BDI-architecture. In James F. Allen, Richard Fikes, and Erik Sandewall, editors, *KR'91: Principles of Knowledge Representation and Reasoning*, pages 473–484. Morgan Kaufmann, San Mateo, California, 1991.

[110] M.K. Reiter and A.D. Rubin. Anonymous web transactions with Crowds. *Communications of the ACM*, 42(2):32–38, February 1999.

[111] M.D. Sadek. A Study in the Logic of Intention. In Bernhard Nebel, Charles Rich, and William Swartout, editors, *KR'92. Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference*, pages 462–473. Morgan Kaufmann, San Mateo, California, 1992.

[112] G. Salton. *Automatic Text Processing: The Transformation, Analysis and Retrieval of Information by Computer.* Addison-Wesley, 1989.

[113] J. Searle. *Speech Acts.* Cambridge Univesity Press, Cambridge, England, 1969.

[114] B. Selman, H. Levesque, and D. Mitchell. GSAT: A new method for solving hard satisfiability problems. In *Proceedings AAAI-92*, pages 440–446, San Jose, CA, USA, 1992.

[115] O. Shehory. A Scalable Agent Location Mechanism. In *Proceedings of ATAL 1999*, pages 162–172, 1999.

[116] O. Shehory and K. Sycara. The RETSINA communicator. In Carles Sierra, Maria Gini, and Jeffrey S. Rosenschein, editors, *Proceedings of the Fourth International Conference on Autonomous Agents*, pages 199–200, Barcelona, Catalonia, Spain, June 2000. ACM Press.

[117] O. Shehory, K. Sycara, P. Chalasani, and S. Jha. Increasing Resource Utilization and Task Performance by Agent Cloning. In Jörg Müller, Munindar P. Singh, and Anand S. Rao, editors, *Proceedings of the 5th International Workshop on Intelligent Agents V : Agent Theories, Architectures, and Languages (ATAL-98)*, volume 1555 of *LNAI*, pages 413–426, Berlin, July 04–07 1998. Springer.

[118] M. Singh. A semantics for speech acts. *Annals of Mathematics and Artificial Intelligence*, 8(1-2):47–71, 1993.

[119] M.P. Singh. A logic of intentions and beliefs. *Journal of Philosophical Logic*, 22:513–544, 1993.

[120] M.P. Singh. Agent Communication Languages: Rethinking the Principles. *IEEE Computer*, 31(12):40–47, December 1998.

[121] N. Singh, M.R. Genesereth, and M. Syed. A Distributed and Anonymous Knowledge Sharing Approach to Software Interoperation. *International Journal of Cooperative Information Systems*, 4(4):339–368, 1995.

[122] N. Skarmeas and K. Clark. Content-Based Routing as the Basis for Intra-Agent Communication. In Jörg Müller, Munindar P. Singh, and Anand S. Rao, editors, *Proceedings of the 5th International Workshop on Intelligent Agents V : Agent Theories, Architectures, and Languages (ATAL-98)*, volume 1555 of *LNAI*, pages 345–362, Berlin, July 04–07 1998. Springer.

[123] S. Soltysiak, T. Ohtani, M. Thint, and Y. Takada. An Agent-Based Intelligent Distributed Information Management System for Internet Resources. Available at http://www.isoc.org/inet2000/cdproceedings/2f/2f_1.htm.

[124] Sun Microsystems, Inc. Java Distributed Event Specification, 1998.

[125] The Semantic Web web site: http://www.semantic-web.org.

[126] K. Sycara, M. Klusch, S. Widoff, and J. Lu. Dynamic Service Matchmaking Among Agents in Open Information Environments. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 28(1):47–53, 1999.

[127] K. Sycara, J. Lu, M. Klusch, and S. Widoff. Matchmaking among Heterogeneous Agents on the Internet. In *Proceedings of the 1999 AAAI Symposium on Intelligent Agents in Cyberspace*, March 1999.

[128] K. Sycara, S. Widoff, M. Klusch, and J. Lu. LARKS: Dynamic Matchmaking Among Heterogeneous Software Agents in Cyberspace. *Autonomous Agents and Multi-Agent Systems*, 5:173–203, 2002.

[129] Y. Takada, T. Mohri, and H. Fujii. Mediators in the Architecture of Future Information Systems. *IEEE Computer*, 25(3):38–49, 1992.

[130] Y. Takada, T. Mohri, and H. Fujii. Multi-agent System for Virtually Integrating Distributed Databases. *FUJITSU Sci. Tech. Journal*, 34(2):245–255, December 1998.

[131] R. Tewari, M. Dahlin, H. Vin, and J. Kay. Beyond Hierarchies: Design Considerations for Distributed Caching on the Internet. In *Proceedings of ICDCS 99*, May 1999.

[132] P. Triantafillou and A. Economides. Subscription Summaries for Scalability and Efficiency in Publish/Subscribe Systems. In *IEEE Workshop on Distributed Event-based Systems*, July 2002.

[133] P. Triantafillou, C. Xiruhaki, M. Koubarakis, and N. Ntarmos. Towards High-Performance Peer-to-Peer Content and Resource Sharing Systems. In *Conference on Innovative Data Systems Research (forthcoming)*, January 5-9 2003.

[134] A. Umar. *Distributed Computing: A Practical Synthesis*. Prentice-Hall, 1993.

[135] F. van Harmelen and I. Horrocks. FAQs on OIL: The Ontology Inference Layer. *Trends and Controversies Column. IEEE Intelligent Systems*, 15(6), November 2000.

[136] W3C. XML Path Language (XPath) 1.0. http://www.w3.org/TR/xpath, 1999.

[137] W3C. XQuery 1.0. http://www.w3.org/TR/xquery, 2002.

[138] G. Wiederhold. Intelligent integration of information. In Peter Buneman and Sushil Jajodia, editors, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, SIGMOD '93, Washington, DC, May 26–28, 1993*, volume 22(2) of *SIGMOD Record (ACM Special Interest Group on Management of Data)*, pages 434–437, New York, NY 10036, USA, 1993. ACM Press.

[139] G. Wiederhold. Value-added Mediation in Large-Scale Information Systems. In R. Meersman and L. Mark, editors, *Proceedings of the 6th IFIP TC-2 Working Conference on Data Semantics (DS-6)*, pages 34–56, 1995.

[140] G. Wiederhold and M.R. Genesereth. The Conceptual Basis for Mediation Services. *IEEE Expert*, 12(5):38–47, 1997.

[141] I.H. Witten, A. Moffat, and T.C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kauffman Publishing, San Francisco, 2nd edition, 1999.

[142] H. Chi Wong and K. Sycara. A Taxonomy of Middle-Agents for the Internet. In *Proceedings of 4th International Conference on Multi Agent Systems (ICMAS-2000)*, Boston, Massachusetts, July 2000.

[143] T.W. Yan and H. Garcia-Molina. Distributed selective dissemination of information. In *Proceedings of the 3rd International Conference on Parallel and Distributed Information Systems (PDIS)*, pages 89–98, 1994.

[144] T.W. Yan and H. Garcia-Molina. Index structures for selective dissemination of information under the boolean model. *ACM Transactions on Database Systems*, 19(2):332–364, 1994.

[145] T.W. Yan and H. Garcia-Molina. The SIFT information dissemination system. *ACM Transactions on Database Systems*, 24(4):529–565, 1999.

[146] C. Zhang, J. Naughton, D. De Witt, Q. Luo, and G. Lohman. On Supporting Containment Queries in Relational Database Management Systems. In *Proceedings ACM SIGMOD International Conference on Management of Data*, 2001.