# Information Alert in Distributed Digital Libraries: The Models, Languages and Architecture of DIAS*

M. Koubarakis, T. Koutris, C. Tryfonopoulos, and P. Raftopoulou

Dept. of Electronic and Computer Engineering
Technical University of Crete
73100 Chania, Crete, Greece
{manolis,koutris,trifon,rautop,}@intelligence.tuc.gr
www.intelligence.tuc.gr/~manolis

**Abstract.** This paper presents DIAS, a distributed alert service for digital libraries, currently under development in project DIET. We first discuss the models and languages for expressing user profiles and notifications. Then we present the data structures, algorithms and protocols that underly the peer-to-peer agent architecture of DIAS.

## 1 Introduction

Users of modern digital libraries can keep themselves up-to-date by searching and browsing their favourite collections, or more conveniently by resorting to an *alert service*. Recently, the participants of project HERMES [15] have argued very convincingly that an alert service which integrates information from a wide variety of information providers can be indispensable to users. It relieves them from the tedious and cumbersome task of searching and browsing, or even from subscribing to individual alert services such as Springer Link Alert[1] or Elsevier Contents Direct[2].

In this paper, we discuss the models, languages and architecture of DIAS, a *D*istributed *I*nformation *A*lert *S*ystem currently under development in the context of the European project DIET [24, 27, 18]. DIAS adopts the basic ideas of project HERMES [15] and extends them in various interesting ways.

Our main technical contributions can be summarized as follows. First, we introduce the peer-to-peer (P2P) agent architecture of DIAS which has been inspired by the event dissemination system SIENA [6]. We also discuss the requirements imposed by this architecture on the data models and languages to

* This work was carried out as part of the DIET (Decentralised Information Ecosystems Technologies) project (IST-1999-10088), within the Universal Information Ecosystems initiative of the Information Society Technology Programme of the European Union.

1 `http://link.springer.de/alert`

2 `http://www.elsevier.nl`

be used for specifying user profiles/queries and notifications.[3] Then we develop formally the data models $\mathcal{WP}$, $\mathcal{AWP}$ and $\mathcal{AWPS}$, and their corresponding languages for specifying queries and notifications. Data model $\mathcal{WP}$ is only briefly presented and more details can be found in [19, 20]. $\mathcal{WP}$ 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 [2, 8, 9] in a significant way, and utilize it in a query language targeted at information alert for distributed digital libraries. 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 [6, 4] by using linguistically motivated concepts such as *word* and not arbitrary strings. This makes $\mathcal{AWP}$ very appropriate in information alert systems for digital libraries[4]. 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 [2]. 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 [29] where documents and queries are represented by free text. The similarity concept of $\mathcal{AWPS}$ is essentially the similarity concept pioneered by IR systems [2], database systems with IR influences (e.g., WHIRL [11]) and more recently by the XML query language ELIXIR [10]. 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 first presentation of model $\mathcal{AWPS}$ is the one given in this paper and [21].

In the second part of our paper, we present the detailed protocols and algorithms of our architecture and discuss its implementation in the context of project DIET [24, 27, 18]. Contrary to project HERMES [15], we develop a distributed information alert service from scratch based on the ideas of SIENA [6] and do not rely on any pre-existing message-oriented middleware.

The rest of the paper is organised as follows. Section **??** introduces the DIAS architecture, and discusses the requirements for data models and languages to be used in this context. Section 3 presents data model $\mathcal{WP}$ and its semantics. Then Sections 4 and 5 build on this foundation and develops the same machinery for data models $\mathcal{AWP}$ and $\mathcal{AWPS}$. Section 6 discusses some interesting details of the DIAS architecture and implementation. Finally, Section 7 gives our conclusions and discusses future work.

[3] In this paper, the terms *query* and *profile* will be used interchangeably. We are in an information alert setting where a profile is simply a *long-standing query*.

[4] Also, for other commercial systems where similar models are supported already for retrieval.

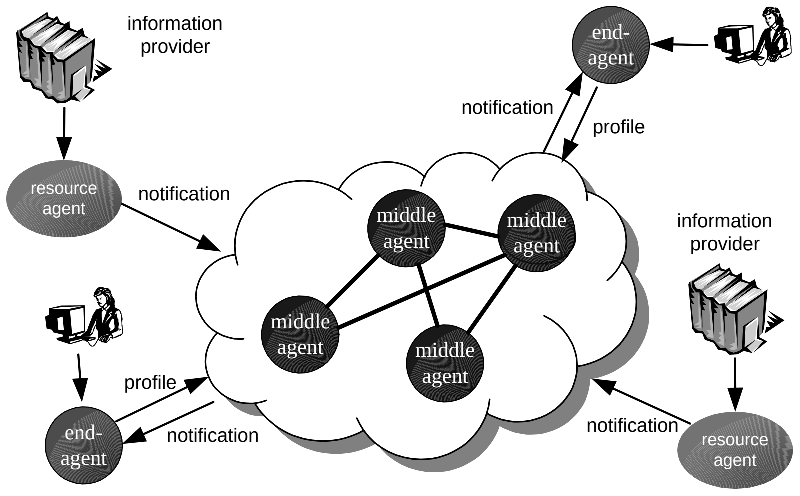

**Fig. 1.** The architecture of DIAS

## 2 DIAS: Architecture and Requirements

A high level view of DIAS is shown in Figure 1. *Resource agents* integrate a number of information providers to produce streams of notifications that are send to some *middle-agent(s).* Resource agents are similar in functionality to observers as defined in project HERMES [15]. Users utilize their *personal agents* to post profiles to some middle-agent(s). The P2P network of middle-agents is the “glue” that makes sure that published notifications 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 notification can take place at a middle-agent that is as close as possible to the origin of the incoming notification. Profile forwarding can be done in a sophisticated way to minimize network traffic e.g., no profiles that are less general than one that has already been processed are actually forwarded.

Most of the concepts of the architecture sketched above have been explicit (or sometimes implicit) in the agent literature for some time (especially, the literature on KQML and subsequent multi-agent systems based on it [16]. 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 [6] where the distributed event dissemination system SIENA is presented.[5] Some core ideas of SIENA have been adopted by DIAS as we will explain in Section 6.

[5] SIENA does not use terminology from the area of agent systems but the connection is obvious.

From the beginning, the design of DIAS has proceeded in a *principled* and *formal* way. With this motivation and the above architecture in mind, we now proceed to discuss our requirements for models and languages to be used in this setting:

1. *Expressivity.* The language for notifications and user profiles must be rich enough to satisfy the demands of users and capabilities of information providers.
2. *Formality.* The syntax and semantics of the proposed models and languages must be defined formally.
3. *Computational efficiency.* The following problems should be defined formally and algorithms must be provided for their efficient solution:
   (a) The *satisfiability problem*: Deciding whether a profile can be satisfied by any incoming notification at all. This functionality is necessary at each middle-agent.
   (b) The *satisfaction* (or *matching*) *problem*: Deciding whether a notification satisfies (or matches) a profile.
   (c) The *filtering problem*: Given a database of profiles $db$ and a notification $n$, find all profiles $q \in db$ that match $n$. 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 alert systems to handle hundreds of thousands or millions of profiles.
   (d) The *entailment* or *subsumption problem*: 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.

Previous research has shown that the above formal perspective is shared by other researchers in the area of information dissemination.[6] Various papers proposing sophisticated filtering algorithms in the area of event dissemination have defined the syntax and semantics of their languages carefully [1, 25, 14]. The developers of the distributed event dissemination system SIENA in particular have carefully defined the syntax and semantics of their language for events and profiles, and have formalised the notions of satisfaction and entailment (called "covers" in [6]). The same has been done for a much more expressive language in [4] where only satisfaction and filtering have been considered. To the best of our knowledge, no work has been done so far on addressing the satisfiability problem.

In the rest of this paper, we concentrate on information alert in distributed digital libraries and define the models $\mathcal{WP}$, $\mathcal{AWP}$ and $\mathcal{AWPS}$ that are suitable for this job. Then we discuss the architecture of DIAS and show how the theoretical concepts discussed above become crucial for its efficient implementation.

## 3 Text Values and Word Patterns

The model $\mathcal{WP}$ assumes that textual information is in the form of *free text* and can be queried by *word patterns* (hence the acronym for the model).

[6] However, to the best of our knowledge, no one has applied this formal perspective in a distributed digital library context.

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 a (finite or 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 $|s|$.

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 generated by the grammar*

$$WP \rightarrow \mathbf{w} \mid \neg WP \mid WP \wedge WP \mid WP \vee WP \mid (WP)$$

*where terminal $\mathbf{w}$ represents a word of $\mathcal{V}$. A proximity-free word pattern will be called* positive *if it does not contain the negation operator.*

*Example 1.* In all the examples of this paper, our vocabulary will be the vocabulary of the English language and will be denoted by $\mathcal{E}$. The following are proximity-free word patterns that might appear in queries of a user of an information alert system interested in articles on constraints:

$$constraint \wedge programming \wedge \neg e\text{-}commerce,$$
$$constraint \wedge (optimisation \vee programming)$$

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 2.* The following are proximity word patterns:

$$constraint \prec_{[0,0]} programming, \quad artificial \prec_{[0,0]} neural \prec_{[0,0]} networks,$$
$$algorithms \prec_{[0,3]} (satisfaction \vee filtering), \quad induced \prec_{[0,\infty)} constraints$$
$$repairing \prec_{[0,10]} querying \prec_{[0,10]} (inconsistent \wedge databases)$$

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 $algorithms \prec_{[0,3]} satisfaction$ denotes that the word "satisfaction" appears after word "algorithms" and at a distance of at least 0 and at most 3 words. The word pattern $constraint \prec_{[0,0]} programming$ denotes that the word "constraint" appears exactly before word "programming" so this is a way to encode the string "constraint programming". 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.

**Definition 4.** *Let $\mathcal{V}$ be a vocabulary. A* word pattern *over vocabulary $\mathcal{V}$ is an expression generated by the grammar*

$$WP \rightarrow PFWP \mid PWP \mid WP \wedge WP \mid WP \vee WP \mid (WP)$$

*where non-terminals $PFWP$ and $PWP$ represent proximity-free and proximity word patterns respectively. A word pattern will be called* positive *if its proximity-free subformulas are positive.*

*Example 3.* The following are word patterns of the most general kind we allow:

$$applications \wedge (constraint \prec_{[0,0]} programming) \wedge \neg e\text{-}commerce,$$
$$algorithms \wedge (complexity \prec_{[0,10]} (satisfaction \wedge filtering)),$$
$$learning \ \wedge ((neural \prec_{[0,0]} networks) \vee \ (neuromorphic \prec_{[0,0]} systems))$$

It is not difficult to see now how to define what it means for a text value $s$ to *satisfy* a word pattern $wp$ (denoted by $s \models wp$). The exact definition is given in [19–21].

*Example 4.* Let $s$ be the following text value:

"*Interaction of constraint programming and local search for optimisation problems*"

The text value $s$ satisfies the following word patterns:

$$local \prec_{[0,0]} search \prec_{[0,5]} optimisation$$
$$(global \vee local) \prec_{[0,5]} search \prec_{[1,1]} optimisation,$$
$$(constraint \wedge programming) \prec_{[0,10]} optimisation \prec_{[0,0]} problems$$

The text value $s$ also satisfies word pattern:

$$optimisation \wedge (constraint \prec_{[0,0]} programming)$$

Finally we define entailment (conversely: subsumption) of two word patterns.

**Definition 5.** *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 *$wp_1$.*

*Example 5.* The word pattern $constraint \wedge programming$ entails $programming$. The word pattern $learning \wedge (artificial \prec_{[0,0]} neural \prec_{[0,0]} networks)$ entails the word pattern $learning \wedge (neural \prec_{[0,0]} networks) \wedge artificial$. The word pattern $algorithms$ subsumes the word pattern $algorithms \wedge satisfaction \wedge filtering$. Similarly, $satisfaction \vee filtering$ subsumes $filtering$.

## 4 An Attribute-Based Data Model and Query Language

Now that we have studied the data model $\mathcal{WP}$,we are ready to define our second data model and query language. Data model $\mathcal{AWP}$ is based on *attributes* or *fields* with finite-length strings as values (in the acronym $\mathcal{AWP}$, the letter $\mathcal{A}$ stands for "attribute"). Strings will be understood as sequences of words as formalized by the model $\mathcal{WP}$ presented earlier. Attributes can be used to encode textual information in a notification (e.g., author, title, abstract of a paper and so on). $\mathcal{AWP}$ is somewhat restrictive since it offers a flat view of a notification, but it will suffice in many cases that arise in digital library environments. A similarly flat view of a notification has successfully been adopted in project HERMES [15].

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

**Definition 6.** *A* notification schema *$\mathcal{N}$ 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 6.* An example of a notification schema for information alert in a digital library is

$$\mathcal{N} = (\{AUTHOR,\ TITLE,\ ABSTRACT\}, \mathcal{E}).$$

**Definition 7.** *Let $\mathcal{N} = (\mathcal{A}, \mathcal{V})$ be a notification schema. A* notification *$n$ over schema $\mathcal{N}$ 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 7.* The following is a notification over the schema of Example 6:

$$\begin{gathered} \{\ (AUTHOR, \text{“}John\ Brown\text{”}), \\ (TITLE, \text{“}Interaction\ of\ constraint\ programming\ and \\ local\ search\ for\ optimisation\ problems\text{”}), \\ (ABSTRACT, \text{“}In\ this\ paper\ we\ show\ that \\ adapting\ constraint\ propagation...\text{”})\ \} \end{gathered}$$

Notice that *only* textual information is allowed in notifications as defined in this paper. In the DIAS implementation to be discussed in Section 6, we eventually plan to support other kinds of information (e.g., numerical, dates, etc.). However, at the moment, we concentrate only on textual information.

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

**Definition 8.** *Let $\mathcal{N} = (\mathcal{A}, \mathcal{V})$ be a notification schema. A* query *over $\mathcal{N}$ is a formula in any of the following forms:*

1. *$A \sqsupset wp$ where $A \in \mathcal{A}$ and $wp$ is a* positive *word pattern over $\mathcal{V}$. The formula $A \sqsupset wp$ 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 $\phi$ is a query containing no proximity word patterns.*
4. *$\phi_1 \vee \phi_2$ where $\phi_1$ and $\phi_2$ are queries.*
5. *$\phi_1 \wedge \phi_2$ where $\phi_1$ and $\phi_2$ are queries.*

The queries in the first two of the above cases are called *atomic*.

*Example 8.* The following are queries over the schema of Example 6:

$$AUTHOR \sqsupset (John \prec_{[0,2]} Smith),$$
$$\neg AUTHOR = \text{“}John\ Smith\text{”} \wedge$$
$$(TITLE \sqsupset (optimisation \wedge (constraint \prec_{[0,2]} programming)))$$

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

**Definition 9.** *Let $\mathcal{N}$ be a notification schema, $n$ a notification over $\mathcal{N}$ and $\phi$ a query over $\mathcal{N}$. The concept of notification $n$* satisfying *query $\phi$ (denoted by $n \models \phi$) is defined as follows:*

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

*Example 9.* The first query of Example 8 is not satisfied by the notification of Example 7 while the second one is satisfied.

Finally we define entailment (conversely: subsumption) of two queries.

**Definition 10.** *Let $\phi_1$ and $\phi_2$ be queries over schema $\mathcal{N}$. Then $\phi_1$* entails *$\phi_2$ (denoted by $\phi_1 \models \phi_2$) iff every notification $n$ over $\mathcal{N}$ that satisfies $\phi_1$ also satisfies $\phi_2$. If $\phi_1$ entails $\phi_2$ then we also say that $\phi_2$* subsumes *$\phi_1$.*

*Example 10.* The query

$(AUTHOR \sqsupset John \prec_{[0,1]} Brown) \wedge (TITLE \sqsupset constraint \wedge programming)$

entails (equivalently: is subsumed by) $TITLE \sqsupset programming$.

## 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 $\sqsupset$. This operator is very useful for queries such as “I am interested in papers written by John Brown” which can be written in $\mathcal{AWP}$ as

$$AUTHOR \sqsupset (John \prec_{[0,0]} Brown)$$

but it might not be very useful for queries “I am interested in papers about the use of local search techniques for the problem of test pattern optimisation”.

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) [2, 23, 28]. 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 $\omega_i$ 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 recently demonstrated by many successful IR systems [2] or database systems adopting ideas from IR (see for example, WHIRL [11]). [7]

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 11.** *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 $\omega_{ij}$.*

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 alert scenarios 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

[7] Sometimes in IR systems (or systems adopting ideas from IR) word *stems*, produced by some stemming algorithm [26], 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 paper, but it should be understood that our current implementation of the ideas of this section utilizes these standard techniques.

computations involving attribute $A$ (the details are given below). $C_A$ can be a collection of recently processed text values as suggested in [29, 15].

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:[8]

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

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 8 with one more case for atomic queries:

– $A \sim_k s$ 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 11.* The following are some queries in $\mathcal{AWPS}$ using the schema of Example 7:

$$TITLE \sim_{0.6} \text{“}Local\ search\ techniques\ for\ constraint\ optimisation\ problems\text{”},$$
$$(AUTHOR \sqsupset (John \prec_{[0,2]} Smith)) \wedge$$
$$(TITLE \sim_{0.9} \text{“}Temporal\ constraint\ programming\text{”}),$$
$$TITLE \sim_{0.9} \text{“}Large\ Scale\ Telecommunication\ Network\ Optimisation\text{”}$$

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 9 with one additional case for the similarity operator:

– If $\phi$ 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 alert setting by [17] and later on adopted by [29]. 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 alert system because very few documents (or even a single one) enter the system at a time, and need to be forwarded to interested users (see the architecture sketched in Figure 1).

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

[8] The IR literature gives us several very closely related ways to define the notions of weight and similarity [2, 23, 28]. 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.

result in very few achieving satisfaction (or even no documents at all). In DIAS, users could 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 [7]. This update functionality has not been implemented in the current version of DIAS.

*Example 12.* The first query of Example 11 is likely to be satisfied by the document of Example 7 (of course, we cannot say for sure until we know the *idf* factors so that the exact weights can be calculated). The second query is not satisfied, since the author specified in the query does not match the document's author. Moreover the third query is unlikely to be satisfied since the only common word between the query and Example 7 is the word "optimisation".

## 6 The Current DIAS Implementation: Details

In the previous sections of this paper, we have defined appropriate models and languages for expressing profiles and notifications in DIAS. For efficiency reasons, the current implementation of DIAS supports only profiles of the form $\psi \wedge \sigma$ where

- $\psi$ is a conjunction of atomic queries of $\mathcal{AWP}$ of the form $A = s$ or $A \sqsupset wp$ where every proximity formula in the word pattern $wp$ contains only words as subformulas.
- $\sigma$ is a conjunction of atomic similarity queries as defined in the model $\mathcal{AWPS}$.

The notion of notification supported currently in DIAS is as defined in Section 4.

The reasons for the above choices in the current DIAS implementation are as follows. In [13], we give precise bounds for the computational complexity of all problems defined in Section **??** for model $\mathcal{AWP}$. Satisfaction can always be decided in polynomial time. For the complete query language of $\mathcal{AWP}$, the satisfiability problem is NP-complete and the entailment problem is coNP-complete. Polynomial cases of these two problems have been identified and the most useful one is the conjunctive case adopted in DIAS. This theoretical analysis will appear in a forthcoming paper.

In the rest of this section we discuss the data structures, algorithms and protocols that regulate how DIAS agents work together so that all produced information is delivered to interested consumers. This is achieved with the combination of the following techniques: sophisticated forwarding of the $\mathcal{AWP}$ part of each subscription by utilizing appropriate networking algorithms [12] and a poset data structure, and very fast indexing techniques for detecting the set of profiles that match an incoming notification. Preliminary evaluation of these techniques by us (and previously by SIENA researchers [6, 5]) lead us to believe that DIAS will be a robust and scalable system.

*Forwarding the $\mathcal{AWP}$ Part of a Subscription.* DIAS follows the lead of SIENA [6, 5] and forwards subscriptions using the *reverse path forwarding algorithm* of [12]. In DIAS this algorithm works as follows. Let us assume that at one end of the information chain, a personal agent $A$ posts a profile $p \equiv \psi \wedge \sigma$ to some middle-agent $M$ ($\psi$ and $\sigma$ are as defined at the beginning of this section). Then, the $\mathcal{AWP}$ part $\psi$ of $p$ is propagated through the middle-agent network so that a *spanning tree* that connects $M$ to all other middle-agents is formed. More precisely, a middle-agent $M$ forwards an $\mathcal{AWP}$ profile $\psi$ only if $\psi$ is coming from a neighbor middle-agent $N$ that is on the *shortest path* connecting the source of $\psi$ with middle-agent $M$. If this condition is satisfied then $M$ can forward $\psi$ to all its neighbors except the one that originally forwarded $\psi$ to $M$.

The above algorithm requires that every middle-agent $M$ knows the identity of the neighbor middle-agent that is the next node on the shortest path from $M$ to every other middle-agent in the network. To keep track of this information, every middle-agent maintains an appropriate routing table that associates a *source* middle-agent to a *neighbor* middle-agent and a *distance* which reflects the latency between nodes in the network. This routing table can be constructed using traditional techniques such as the asynchronous distributed version of Bellman-Ford's all-pairs shortest-path algorithm [3].

*The Profile Poset.* At each middle-agent, the propagation of a newly arrived $\mathcal{AWP}$ profile $\psi$ is implemented by treating $\psi$ as a subscription to neighboring middle-agents. This propagation step is optimized (network traffic is minimized) by allowing it only towards those agents that have not received already a more general profile $\phi$ (i.e., $\psi \models \phi$). To achieve this, each agent maintains a *partially ordered set* (called the *profile poset*) that keeps track of the subsumption relations among the $\mathcal{AWP}$ profiles posted to it.[9] This is possible because the relation of subsumption in $\mathcal{AWP}$ (see Section 4) is reflexive, anti-symmetric and transitive i.e., a *(weak) partial order* [22].

As $\mathcal{AWP}$ profiles arrive at a middle-agent from neighboring middle-agents, they are inserted into the local profile poset. If for two profiles $\psi$ and $\phi$ of a given poset $P$, $\psi \models \phi$, then $\phi$ will be an *ancestor* of $\psi$ (and conversely $\psi$ will be a *descendant* of $\phi$) in $P$. Although the relations ancestor/descendant are transitive relations, the poset maintains only *immediate* ancestor/descendant relations. Figure 2 shows an example of a profile poset.

To facilitate the routing of subscriptions and notifications, every profile $\psi$ in a poset is associated with two sets: a set of *subscribers* that contains the identities of other middle-agents that have subscribed with profile $\psi$, and a set of *forwardees* that contains the addresses of neighbor middle-agents where profile $\psi$ has been forwarded.

The insertion of an $\mathcal{AWP}$ profile $\psi$ in the poset of a middle-agent $M$ is performed according to the following rules:

[9] See [22] for posets and related definitions.

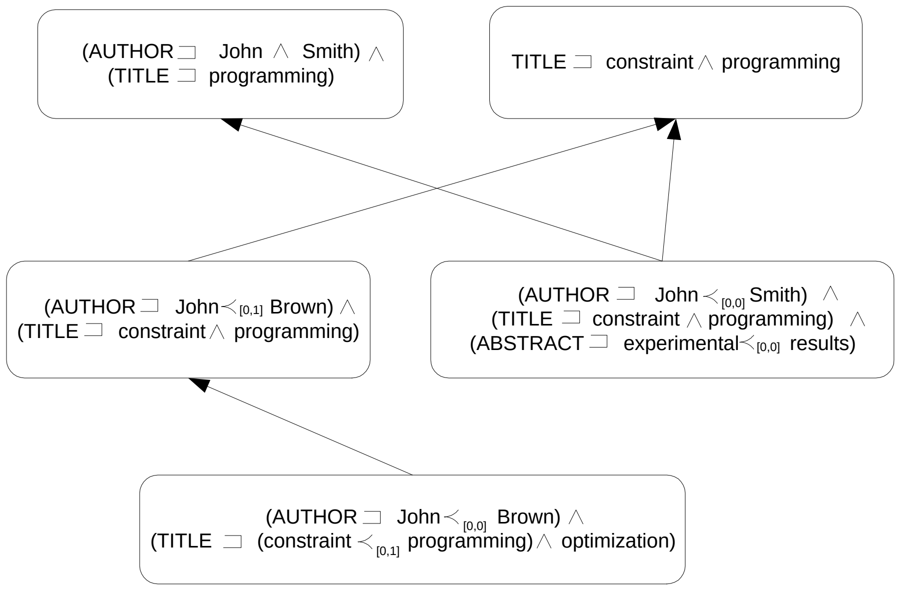
(AUTHOR ⊒ John ∧ Smith) ∧ (TITLE ⊒ programming)
TITLE ⊒ constraint ∧ programming
(AUTHOR ⊒ John ≺[0,1] Brown) ∧ (TITLE ⊒ constraint ∧ programming)
(AUTHOR ⊒ John ≺[0,0] Smith) ∧ (TITLE ⊒ constraint ∧ programming) ∧ (ABSTRACT ⊒ experimental ≺[0,0] results)
(AUTHOR ⊒ John ≺[0,0] Brown) ∧ (TITLE ⊒ (constraint ≺[0,1] programming) ∧ optimization)

**Fig. 2.** A profile poset

1. If a profile subscription $\phi$ that subsumes $\psi$ is already present in the poset, and has the source $S$ of the profile among its subscribers, the insertion ends without updating the poset structure.
2. If $\psi$ exists in the poset, and $S$ is not among its subscribers, $M$ simply inserts the current subscriber $S$ in the set of subscribers of $\psi$.
3. If $\psi$ does not exist in the poset, the two (possibly empty) sets $f$ and $f'$, representing the immediate ancestors and the immediate descendants of $\psi$, are computed, and $\psi$ is inserted as a new node between $f$ and $f'$.

In the latter two cases, $M$ also removes $S$ from all the subscriptions in the poset that are subsumed by $\psi$. If this process leaves a subscription with an empty set of subscribers then this subscription is removed from the poset.

After the poset is updated, $M$ forwards $\psi$ to all its neighbors except to those that $M$ has already forwarded a profile that subsumes $\psi$ (and of course except the neighbor that originally sent this profile to $M$). The complexity of this algorithm for inserting a profile in a poset is polynomial (the exact bound is given in [13]). Note that $M$ could also forward the similarity part $\sigma$ of a profile if an appropriate entailment relation had been defined for model $\mathcal{AWPS}$. This is currently an open problem for us and similarity queries $\sigma$ participate only in filtering notifications as explained below.

*Processing Notifications.* If a resource agent $R$ publishes a notification $n$ that matches an existing subscription $\psi$, $n$ is routed towards the middle-agent $M$ where $\psi$ originated by following the *reverse path* defined by the earlier propagation of $\psi$. The poset structure discussed above can be used to solve this filtering problem and this is what it is done in SIENA [6]. We have not been satisfied by the performance of such poset structures for filtering and we follow a different approach. We use specialized filtering algorithms that utilize indices over the database of profiles at each middle-agent. When an incoming notification $n$ arrives, the index is used to retrieve all profiles satisfying $n$ and their subscribers rapidly. Scalable filtering algorithms of this kind that can deal with millions of $\mathcal{AWP}$ profiles are discussed in [13] and will appear in a forthcoming paper.

## 7 Conclusions

In this paper we presented DIAS, a distributed alert service for digital libraries, currently under development in project DIET. We first discussed the models and languages for expressing user profiles and notifications culminating in the development of model $\mathcal{AWPS}$. Then we presented the data structures, algorithms and protocols that underlie the P2P agent architecture of DIAS. $\mathcal{AWPS}$ is an expressive formal model especially targeted for information alert in distributed digital libraries. However, it is fair to say that in our quest for formality and expressive power, we have overlooked the issue of adaptation of user profiles and how this would affect the design and algorithms of DIAS. In this area there is a lot of interesting IR research (e.g., see the work in the TREC filtering track[10]) but it is an open problem how such ideas can be used in distributed information alert systems such as DIAS.

Our current research concentrates on evaluating analytically and experimentally the current DIAS architecture and comparing it with other alternatives inspired by current research in P2P systems. In this way, we expect that the informal claims we made in Section 6 about scalability and robustness of DIAS will be substantiated.

**Acknowledgements**

This work was carried out as part of the DIET (Decentralised Information Ecosystems Technologies) project (IST-1999-10088), within the Universal Information Ecosystems initiative of the Information Society Technology Programme of the European Union. We would like to thank the other participants in the DIET project, from Departmento de Teoria de Senal y Comunicaciones, Universidad Carlos III de Madrid, the Intelligent Systems Laboratory, BTexact Technologies and the Intelligent and Simulation Systems Department, DFKI, for their comments and contributions. We would also like to thank the reviewers for their very constructive criticisms.

[10] `http://trec.nist.gov`

## References

1. M. Altinel and M.J. Franklin. Efficient filtering of XML documents for selective dissemination of information. In *Proceedings of the 26th VLDB Conference*, 2000.
2. R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, 1999.
3. D. Bertsekas and R. Gallager. *Data Networks*. Prentice Hall, 1987.
4. 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.
5. A. Carzaniga. *Architectures for an Event Notification Service Scalable to Wide-area Networks*. PhD thesis, Politecnico di Milano Italy, 1998.
6. 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.
7. U. Cetintemel, M.J. Franklin, and C.L. Giles. Self-adaptive user profiles for large-scale data delivery. In *ICDE*, pages 622–633, 2000.
8. 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.
9. 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.
10. T. T. Chinenyanga and N. Kushmerick. Expressive retrieval from XML documents. In *Proceedings of SIGIR'01*, September 2001.
11. William W. Cohen. WHIRL: A word-based information representation language. *Artificial Intelligence*, 118(1-2):163–196, 2000.
12. Y. Dalal and R. Metcalfe. Reverse Path Forwarding of Broadcast Packets. *Communications of the ACM*, 21(12):1040–1048, 1978.
13. M. Koubarakis et. al. Project DIET Deliverable 7 (Information Brokering), December 2001.
14. 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.
15. 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.
16. 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.
17. 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.
18. A. Galardo-Antolin, A. Navia-Vasquez, H.Y. Molina-Bulla, A.B. Rodriquez-Gonzalez, F.J. Valvarde-Albacete, A.R. Figueiras-Vidal, T. Koutris, A. Xiruhaki, and M. Koubarakis. I-Gaia: an Information Processing Layer for the DIET Platform . In *Proceedings of the 1st International Joint Conference on Autonomous Agents & Multiagent Systems (AAMAS 2002)*, September 15–19 2002.

19. 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).
20. 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.
21. P. Raftopoulou M. Koubarakis, C. Tryfonopoulos 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. forthcoming.
22. Z. Manna and R. Waldinger. *The Logical Basis of Computer Programming*, volume 1. Addison Wesley, 1985.
23. C.D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing.* The MIT Press, Cambridge, Massachusetts, 1999.
24. P. Marrow, M. Koubarakis, R.H. van Lengen, F. Valverde-Albacete, E. Bonsma, J. Cid-Suerio, A.R. Figueiras-Vidal, A. Gallardo-Antolin, C. Hoile, T. Koutris, H. Molina-Bulla, A. Navia-Vazquez, P. Raftopoulou, N. Skarmeas, C. Tryfonopoulos, F. Wang, and C. Xiruhaki. Agents in Decentralised Information Ecosystems: The DIET Approach. In M. Schroeder and K. Stathis, editors, *Proceedings of the AISB'01 Symposium on Information Agents for Electronic Commerce, AISB'01 Convention*, pages 109–117, University of York, United Kingdom, March 2001.
25. J. Pereira, F. Fabret, F. Llirbat, and D. Shasha. Efficient matching for web-based publish/subscribe systems. In *Proceedings of COOPIS-2000*, 2000.
26. M.F. Porter. An Algorithm for Suffix Striping. *Program*, 14(3):130–137, 1980.
27. F. Wang. Self-organising Communities Formed by Middle Agents. In *Proceedings of the 1st International Joint Conference on Autonomous Agents & Multiagent Systems (AAMAS 2002)*, September 15–19 2002.
28. 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.
29. T.W. Yan and H. Garcia-Molina. The SIFT information dissemination system. *ACM Transactions on Database Systems*, 24(4):529–565, 1999.