

Notification and Recommendation Services for Web Communities *

H. Belhaj Frej, P. Rigaux and N. Spyrtos

December 10, 2004

Abstract

We propose a formal framework in which notification and recommendation services can be defined for a community of users sharing a collection of documents. We assume that the documents reside at the repositories of their providers and that a “mediator”, or digital library, provides the (virtual) integration of all repositories, by indexing the documents using a hierarchy of terms. A provider willing to share a document must register it at the library, providing a document identifier (e.g., a URI) and a document description. The description of a document is seen as a set of terms from the term hierarchy, and so is seen the profile of a user. Notification/recommendation of a document to a user is based upon matching the document description to the user profile. The paper proposes a method for determining the set of users to be notified when a document is inserted, deleted or modified at the library.

1 Introduction

In this paper, we consider a network of document providers and document consumers, willing to share their documents with other providers and/or consumers. We assume that each document resides at the local repository of its provider, so all providers’ repositories, collectively, can be seen as a database of documents spread all over the network. A typical example is that of an e-learning network, where authors of teaching material are the document providers, and learners are the document consumers.

In the scenario that we consider, the distinction between document providers and document consumers is actually blurred. Indeed, a document provider may use documents available in the collection as parts, to compose a new, more complex document. In such a case, the document provider becomes a document consumer. As a consequence, hereafter, we shall use the term user, to mean document provider or document consumer, indifferently.

The activities of the network are supported by a digital library which acts as a mediator, indexing all shareable documents so that users can access them transparently. To this effect, the library offers a number of services, among which the following basic services :

- *Maintenance Services* allow a user to add a new document in the collection, remove a document from the collection, or modify a document already in the collection (provided that the user is the “owner” of that document).
- *Querying Services* allow users to query the document collection in search of documents of interest. Typically, a user accesses documents either to use them for learning purposes, or to re-use them as building blocks when composing new documents.
- *Personalization Services* provide customized documents to users, based on their profiles. In particular, the digital library offers notification and recommendation services. These two services will be the focus of the present paper.

*Research supported by the EU DELOS Network of Excellence in Digital Libraries and the EU IST Project (Self eLearning Networks), IST-2001-39045.

All these services are based on *document descriptions*. A document can be described along various dimensions, such as Author, Language, Editor, Year, Topic, and so on. In this paper, we focus on only one dimension, the Topic dimension. As a consequence, hereafter, we use the term “description” to mean “topic description”.

In this paper we follow the approach of [15], and assume that a description is just a set of keywords, or terms from a tree-structured taxonomy of terms. There are today a number of standard taxonomies for describing documents, such as the ACM Computing Classification System [1]. We assume that the digital library operates from a (fixed) taxonomy, to which all users adhere. As in [15], we also assume that when a user wants to add a new document to the library, he submits two items:

1. a document identifier (e.g., the URI of the document);
2. a description of the document.

These two pieces of information allow the library to provide the basic services mentioned earlier.

Related work

In a typical digital library environment, there are a large number of documents stored in the library and a large number of users accessing the library. As a consequence, the basic services offered by the library (e.g., search for documents of interest) may exhibit poor performance. One way to cope with this problem is to notify, or alert users when new documents of interest to them are added to the library, or recommend new documents to them. Indeed, a notification/recommendation service can help not only in reducing the number of accesses to the library by its users, but also in increasing their satisfaction. Of course, such a service relies on knowledge of user interests by the library, and such knowledge is expressed in what is usually called the user profile. The main contribution of this paper is the modelling of a notification service for digital libraries and the design of an algorithm that optimizes the search of users to be notified when new documents are registered in the library. The interest in notification and recommendation services grew rapidly in the recent years, mainly due to various Web applications.

A notification, or alert system is composed of two parts : the detection of an update event in the Digital library [6, 7, 8] and the definition of the profile [3, 10, 11].

The detection of an event is a delicate task because it implies several sources that are external to the notification system. The work of [9] tries to define a unified model for alarm services on the Web. This model includes three actors: the information source, the alarm module and the user. However, the sources are considered to be independent and they are not obliged to inform the alert system in case of an update event. It is the alert system itself that looks for this information. In our approach, all document providers are required to register every update event (insertion, deletion or modification), concerning any shareable document, in a common place, namely the directory of the library. As a result, we don't need a special module for the detection of update events.

As for the user profile, on which the alert service is based, it can be defined in many different ways, as discussed in [3]. It can include many aspects like what information does the user need (document content, language of the document, author, publisher...), how and when to deliver this information, ... In the SeLeNe project [10], when the user subscribes to a library, he has to supply some profile information. This information can be mandatory, such as personal identification, address, age, etc, or optional such as his background (certifications, qualifications, licences), information about the goal of his subscription, and so on. Of course, the user could choose not to supply any personal information, but the more the system knows the user, the better the quality of notification will be.

In this paper, we restrict our attention to the topic dimension of a document, i.e., we assume that the user profile consists of a set of terms from the library taxonomy, indicating the users preferences as to the topics he prefers. This is admittedly an over simplified assumption. However, our goal in this paper is to study notification algorithms, rather than the nature of user profiles.

We note that, from the query language point of view, the user profile that we consider here can be seen as a conjunctive query that is continuously asked to the library.

The processing of continuous queries is mentioned in [4, 12, 14, 13]. In [4] for example, a scalable continuous query system is discussed. This system supports a large number of users and offers a technique that optimizes query answering by grouping queries that are similar according to their signature and defining a same execution plan for all of them.

In our approach we don't need such processing on the "profile queries". We just classify the profiles in a hierarchical way according to the topics the users are interested in. This makes it possible to find directly the set of all users who are interested in an update (and only those users).

The remaining of this paper is organized as follows. In Section 2, we first recall some preliminary definitions from [15] regarding taxonomies and document descriptions, and then introduce maintenance and querying services. In Section 3, we present the personalization service and we define the basic concepts of our model, namely *profiles* and *events*. Next we focus on the *notification* process which informs users whenever an event matches their profiles. This process might be costly in a setting where many users work over a large digital libraries. In Section 4, we develop an efficient technique that optimizes the retrieval of the set of users who must be notified whenever an event occurs. Section 5 outlines some concluding remarks and suggestions for future research.

2 Preliminary Definitions

As we mentioned in the introduction, document content descriptions are built based on a controlled vocabulary, or *taxonomy*, to which all providers adhere. A taxonomy consists of a set of terms together with a subsumption relation between terms. An example of a taxonomy is the well known ACM Computing Classification System. [1]. Figure 1 shows an example of a taxonomy.

Definition 1 (Taxonomy) *A taxonomy is a pair (T, \preceq) where T is a terminology, i.e., a finite and non-empty set of names, or terms, and \preceq is a reflexive and transitive relation over T , called subsumption.*

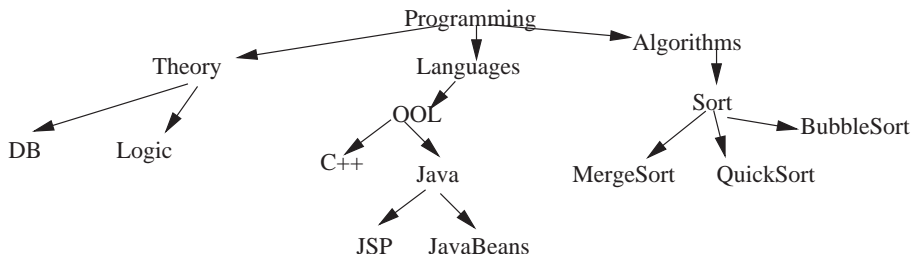


Figure 1: A taxonomy

In order to make a document sharable, we must provide a description of the content, so that users can judge whether the document in question matches their needs. We define such a description to be just a set of terms from the taxonomy. For example, if the document contains the quick sort algorithm written in Java then we can use the terms `QuickSort` and `Java` to describe its content, and the set of terms $\{\text{QuickSort}, \text{Java}\}$ is then a description of the document.

Definition 2 (Description) *Given a taxonomy (T, \preceq) we call description in T any set of terms from T .*

A description D can be redundant if some of the terms it contains are subsumed by other terms. For example, the description $D = \{\text{QuickSort}, \text{Java}, \text{Sort}\}$ is redundant, as `QuickSort` is subsumed by `Sort`. If we remove either `Sort` or `QuickSort` then we obtain a non-redundant description: either $D_1 = \{\text{QuickSort}, \text{Java}\}$ or $D_2 = \{\text{Sort}, \text{Java}\}$, respectively. D_1 is the

minimal reduction of D because we removed all but the minimal terms from D whereas D_2 is the maximal reduction of D . We shall limit our attention to minimal reduction, where intuitively, the reduced description describes as precisely and as economically as possible the LO content.

As we mentioned in the introduction, document identifiers and their associated descriptions constitute the premises for providing some basic services. So let us see first how maintenance services and query services are defined.

Maintenance Services

To add or register a document, its provider must submit to the library a document identifier id and a description D (id may be the URI where the document can be accessed); the library then stores a pair (id, t) for each term t in D . We shall refer to the set of all stored pairs (id, t) , for all documents registered currently at the library, as the *document directory* of the library.

In order to remove a document, its provider must submit to the library the document identifier id ; the library then removes every pair (id, t) . Finally the modification of a document's description, is done by submitting the document identifier id along with its new description D' . The library then performs a removal of id followed by the registration of id with its new description. A document modification can occur when the "owner" of the document has changed its content and feels that its description should change as well.

Querying Services

A query over (T, \preceq) is any boolean combination of terms from T :

$$q ::= t | q \wedge q' | q \vee q' | q \wedge \neg q' | (q) | \epsilon, \text{ where } \epsilon \text{ is the empty query}$$

To define the answer to a query we need the notion of extension of a term t , denoted $Ext(t)$, and defined as the set of documents id such that (id, t) is in the document directory. With this notion at hand, the answer to a query q , denoted by $ans(q)$, is defined recursively as follows:

Case 1: q is a single term t , i.e., $q = t$, then $ans(t) = \bigcup \{Ext(s) | s \preceq t\}$

Case 2: q is a general query

$ans(q) =$
begin
 if $q = q_1 \wedge q_2$, $ans(q) = ans(q_1) \cap ans(q_2)$
 if $q = q_1 \vee q_2$, $ans(q) = ans(q_1) \cup ans(q_2)$
 if $q = q_1 \wedge \neg q_2$, $ans(q) = ans(q_1) \setminus ans(q_2)$
end

Case 3: q is the empty query

$ans(\epsilon) = \emptyset$

3 Personalization Services

There are several services under this heading. However, in this paper, we focus on two personalization services, the notification service and the recommendation service. Actually, these two services have the same goal: if a document with description D is added, removed or modified, then the library must inform all those users for which D matches the user profile.

However, the means by which these two services reach their goal differ in a significant way. In the notification service the profile is defined and maintained by the user (possibly with the assistance of the library). As a consequence, the user is aware of the existence of a profile according to which notification takes place. In the recommendation service, the user is not aware of the existence of a profile: the library analyzes the usage history of each user to define and maintain a user profile, without the user being aware of its existence. The library then recommends a document to the user if the document description matches the user profile.

In both cases we are faced with the same underlying decision problem: if a document is added, removed or modified through a maintenance service, find the users that the system has to alert. We propose in the following a simple formal framework in which this problem can be stated and solved.

The User Profile

Intuitively, a user profile is a statement by the user as to what are his preferences in terms of document content. As such, a profile can be defined by a query over a taxonomy (T, \preceq) , using the query language seen earlier. Consider for example the following query:

$$(OOL \wedge \text{Sorting} \wedge \neg DB) \vee \text{Theory}$$

This query, seen as a user profile, expresses the fact that the user is interested in documents dealing with either object oriented languages and sorting but not databases, or in documents dealing with theory. In the present paper, we focus our attention to profiles that can be expressed as *conjunctive* queries without negation.

Definition 3 (Profile) *A user profile P is a conjunctive query over T .*

Since the profile P of a user u is defined as a query, the set of documents stored in the library that interest u is the query result, $ans(P)$. We assume that the system maintains a *user directory* \mathcal{U} , i.e., a set of pairs (u, P) where u is a user and P the profile of u . For convenience, we shall often use $u.P$ to denote the profile P of a user u .

4 The notification process

The *notification* process consists in alerting users whenever the description of a document that matches their profile is updated in the library. We call *event* such an update. Intuitively, given an event e , any user u whose profile matches e must be notified. This is more precisely defined as follows:

Definition 4 (Notification) *Let $e(id, D)$ be an event affecting the document id with description D . Then $notify(e)$ is the set of users defined as follows:*

$$notify(e) = \{u \in \mathcal{U} \mid id \in ans(u.P)\}$$

Computing $notify(e)$ is an interesting and original problem. Indeed, the traditional approach consists in computing the answer to a given query q whereas we must here find the set of queries whose result is affected by an update operation. The problem can be related of course to *continuous queries*, i.e., queries whose result must be maintained during a given (and possibly unbounded) period of time [4, 12, 14].

Example 1 *Consider the following example:*

- a document id with description $D = \{Java, QuickSort, Theory\}$
- five users u_1, u_2, u_3, u_4, u_5 with the following profiles:
 1. $u_1.P = Java \wedge Sort$
 2. $u_2.P = OOL \wedge Sort \wedge Theory$
 3. $u_3.P = Languages \wedge QuickSort$
 4. $u_4.P = QuickSort \wedge Theory$
 5. $u_5.p = Java \wedge Theory$

Then any changes on id , for $i = 1, \dots, 5$ must be notified to u_i .

There is a straightforward algorithm to compute $notify(e)$: scan the user directory \mathcal{U} , and check for each user u whether the modified document belongs to the answer of $u.P$. Unfortunately this solution is likely to be very costly in the case that we have a large number of users in a library supporting a high ratio of updates.

This trivial algorithm is optimal in the worse case because *all* the users might be interested in an event. However, in practice, one might expect that only a few users will be interested by an event, and we would like to devise an approach which permits to access directly these users. The next section develop a more sophisticated and efficient technique for users notification.

The profiles graph

The approach advocated here relies on the following remark: in many cases we can determine that if a user u is to be notified after an event e , then several other users whose profile is “more general” than $u.P$, must be notified as well. Moreover this dependency between profiles can be determined solely from the content of the user directory. For instance, in the above example, any event e that triggers a notification to u_2 will trigger a notification to u_1 . We explore precisely this intuition, show its correctness, and outline algorithms.

In order to make explicit the relationship among users, we represent a profile by a set of terms. Note that this is only possible because a profile is a conjunctive query, and therefore no ambiguity arises from this representation. For example, the profile $OOL \wedge Sorting \wedge Theory$ is represented by $\{OOL, Sorting, Theory\}$. Next we define the following relation, called *Refinement Relation*.

Definition 5 (Refinement Relation) Let P_1 and P_2 be two profiles. We say that P_1 is finer than P_2 , denoted $P_1 \sqsubseteq P_2$, iff $\forall t \in P_2, \exists t' \in P_1 / t' \preceq t$.

In other words, P_1 is finer than P_2 if every term of P_1 subsumes some term of P_2 . It is easily shown that \sqsubseteq is a partial order. This allows to organise the user directory as follows:

- build the directed acyclic graph $PG = (\mathcal{P}, \sqsubseteq)$, where \mathcal{P} is the set of profiles;
- associate to each profile $P \in \mathcal{P}$ the set \mathcal{U}_P of users u such that $u.P = P$

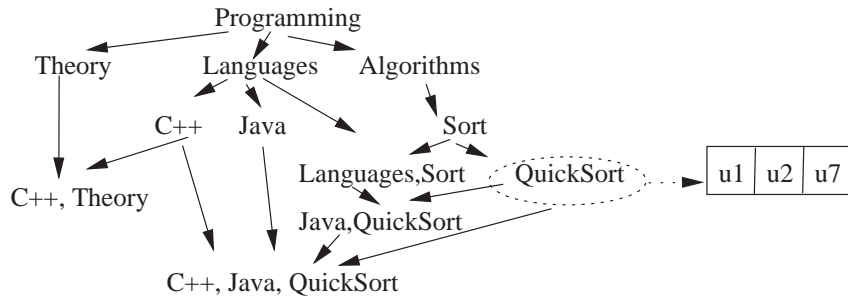


Figure 2: The Profiles graph

PG is the *profiles graph* (see Figure 2). The refinement relation enjoys the following property.

Proposition 1 Let $e(id, D)$ be an event. Then, if a profile P matches e , so does any P' such that $P \sqsubseteq P'$.

If one knows a profile is matched by an event e , then this is true as well for all its ancestors in the graph $(\mathcal{P}, \sqsubseteq)$. Moreover, one can show that $(\mathcal{P}, \sqsubseteq)$ is a lattice [15]. This allows to state the following result:

Proposition 2 *Let e be an event. Then there exists a unique profile P_e such that*

$$\bigcup_{P \in \text{ancestors}(P_e)} \mathcal{U}_P = \text{notify}(e)$$

Indeed, if \mathcal{P}_e is the set of profiles that match e , there exists a greatest lower bound (glb) P_e of \mathcal{P}_e . Finding P_e and its ancestors in the profiles graph allows to retrieve *exactly* the set of users who must be notified.

Scanning the profiles graph

Using the refinement relation just defined, we can now design an algorithm that scans the profiles graph in order to retrieve the set of users to be alerted when a document is inserted, deleted or modified at the library. We envisage several approaches, among which the following are noteworthy:

- maintain the lattice of the profiles graph as users register or unregister their profiles;
- encode the nodes of the graph in such a way that (i) each node can be directly retrieved given a profile encoding and (ii) the ancestors of a profile can be easily constructed.

We briefly describe here the second approach which is currently under implementation. Our encoding relies on an extension of the labelling scheme of the taxonomy tree presented in [2] and further investigated by [5]. In this labelling any node in the taxonomy tree is identified by its position with respect to the parent node. The label is obtained by concatenating the label of the parent and the position. For example, if 1 is the label of the node **Programming**, then 1.1, 1.2 and 1.3 are respectively the labels of its son nodes **Theory**, **Languages** and **Algorithms** (see Figure 3).

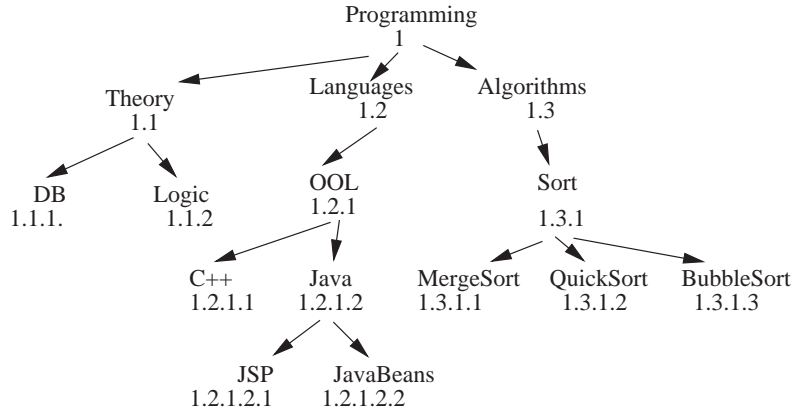


Figure 3: The taxonomy labelling

A profile P is labelled as follows:

- the terms in P are sorted wrt the lexicographic order of their labels;
- given the sorted labels $[l_1, l_2, \dots, l_n]$ the label of P is the string $l_1/l_2/\dots/l_n$.

For example, the profile $\{\text{Theory}, \text{Algorithms}\}$ is coded "1.1/1.3" (see Figure 4). The order of the terms labels in the labelling of P is important because it helps to reduce the number of computations required to obtain the ancestors of P .

Each node corresponds to a bucket storing the list of users who registered the node's profile. All the buckets of the existing profiles are themselves organized in a hash table such that the hash key is the label of each node. This allows to find efficiently a node, given a profile description.

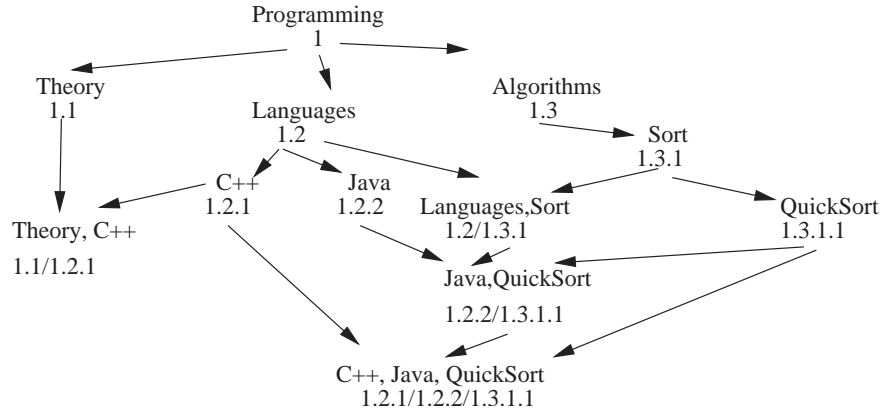


Figure 4: An example of the profiles labels

Next, given the label of a profile P , an algorithm *ancestors* enumerates all the ancestors of P in the profiles graph using an inductive approach based on the number of terms in P . The induction can be stated as follows (for simplicity we blur the distinction between a term and its label):

1. when $P = \{t_1\}$, $ancestors(P)$ is the set of ancestors of t_1 in the taxonomy tree;
2. when $P = \{t_1, \dots, t_n\}$ with $n > 1$, $ancestors(P) = ancestors(\{t_1\}) \bullet ancestors(\{t_2, \dots, t_n\})$

where \bullet is the label concatenation operator that takes two sets S_1 and S_2 of labels, and returns all the reduced labels obtained by concatenating a label from S_1 with a label from S_2 .

Algorithm LABELCONCAT

Input: L_1, L_2 , two sets of labels.

Output: $L_1 \bullet L_2$.

```

begin
  result =  $\emptyset$ 
  for each label  $l_1$  in  $L_1$  do
    for each label  $l_2$  in  $L_2$  do
      if  $(l_1 < l_2)$  then result  $\leftarrow l_1/l_2$ 
      else result  $\leftarrow$  result  $\cup l_1$ 
    end do
  end do
  return result
end

```

Whenever an event $e(id, D)$ is raised, the notification is processed as follows:

1. first initialize the research with the profile composed of the terms in D ; find the corresponding bucket and notify the users (if any);
2. next scan all the ancestors of D ; for each ancestor find the corresponding bucket and notify the users;

Example 2 Let $e(id, D)$ be an event with $D = \{OOL, Sort\}$. The search is initialized with the profile label $1.2.1/1.3.1$ (see Figure 3). The users of the buckets labelled with the following codes must then be notified: $\{1, 1.2, 1.2.1, 1.3, 1.3.1, 1.2/1.3, 1.2/1.3.1, 1.2.1/1.3, 1.2.1/1.3.1\}$

5 Concluding remarks

In this paper we proposed a formalization of the notification process in a dynamic digital library (DL). The central tool of our model is a simple tree-based taxonomy which is used both for documents description and for the query language definition. A profile is a conjunctive query, and the set of profiles registered by the DL can be organized in a graph which can be conveniently used for the notification process.

We are currently implementing two notification algorithms based on this approach. The first one, briefly described in what precedes, encodes the nodes of the profiles graph and uses a simple algorithm to compute the ancestors of a profile. The second approach aims at maintaining the edges of the profiles graph during registration and unregistration of users. From these edges, the graph can be directly scanned without having to compute, at run time (i.e., when an event is received), the label of the parent of a node.

An interesting perspective relates to the case where each user has his own taxonomy and uses it to query the digital library and to express his profile preferences. Finally the proposed approach can be extended to recommendations, using a module that analyses the usage history of each users and automatically constructs his profile.

References

- [1] The acm computing classification system, 1999. www.acm.org/class.
- [2] Rakesh Agrawal, Alexander Borgida, and H. V. Jagadish. Efficient management of transitive relationships in large data and knowledge bases. In *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, 1989.
- [3] Giuseppe Amato and Umberto Straccia. User profile modeling and applications to digital libraries. In *Proceedings ECDL European Conference on Research and Advanced Technology for Digital Librarie*, 1999.
- [4] Jianjun Chen, David J. DeWitt, Feng Tian, and Yuan Wang. NiagaraCQ: a scalable continuous query system for Internet databases. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, 2000.
- [5] V. Christophides, D.Plexousakis, M.Scholl, and S. Tourtounis. On labeling schemes for the semantic web. In *Proceedings of the WWW International Conference on World Wide Web*, 2003.
- [6] D. Faensen, L. Faulstich, H. Schweppe, A. Hinze, and A. Steidinger. Hermes: a notification service for digital libraries. In *Proceedings of the ACM/IEEE JCDL Joint Conference on Digital Libraries*, 2001.
- [7] D. Faensen, A. Hinze, and H. Schweppe. Alerting in a digital library environment – do channels meet the requirements? In *Proceedings of the Second European Conference on Research and Advanced Technology for Digital Libraries*, 1998.
- [8] Annika Hinze. Does alerting have special requirements for query languages? In *Workshop Grundlagen von Datenbanken*, 2001. <http://citeseer.ist.psu.edu/hinze01does.html>.
- [9] Annika Hinze and Daniel Faensen. A unified model of internet scale alerting services. In *Proceedings of the ICSC International Computer Science Conference on Internet Applications*, 1999.
- [10] K. Keenoy, M. Levene, and D. Peterson. Personnalisation and trails in self e-learning networks. Technical report, SeLeNe, 2003. <http://www.dcs.bbk.ac.uk/ap/projects/selene/reports/>.

- [11] K.Keenoy, G.Papamarkos, A.Poulovassilis, M.Levenue, D.Peterson, P.T.Wood, and G.Loizou. Self e-learning networks – functionality, user requirements and exploitation scenarios. Technical report, SeLeNe, 2003. <http://www.dcs.bbk.ac.uk/ap/projects/selene/reports>.
- [12] Ling Liu, Calton Pu, and Wei Tang. Continual queries for internet scale event-driven information delivery. *IEEE Transactions on Knowledge and Data Engineering*, 11(4):610–628, 1999.
- [13] Ling Liu, Calton Pu, and Wei Tang. Continual queries for internet scale event-driven information delivery. *Knowledge and Data Engineering*, 11(4):610–628, 1999.
- [14] Sam Madden, Mehul A. Shah, Joseph M. Hellerstein, and Vijayshankar Raman. Continuously adaptive continuous queries over streams. In *Proceedings of the SIGMOD International Conference on Management of Data*, 2002.
- [15] P. Rigaux and N. Spyrtatos. Metadata Management and Learning Object Composition in a Self e-Learning Network. In *Intl. Workshop on Information Search, Integration and Personalization*, 2003.