Access Control for Sharing Semantic Data across Desktops

* Juri L. De Coi, Ekaterini Ioannou, Arne Koesling, Wolfgang Nejdl, Daniel Olmedilla

L3S Research Center/Leibniz Universität Hannover
Appelstr. 9a, 30167 Hanover, Germany
{lastname}@L3S.de

Abstract. Personal Information Management (PIM) systems aim to provide convenient access to all data and metadata on a desktop to the user itself as well as the co-workers. Obviously, sharing desktop data with co-workers raises privacy and access control issues which have to be addressed. In this paper we discuss these issues, and present appropriate solutions. In line with the architecture of current PIM systems [8, 2, 11, 15], our solutions cover all semantic data shared in such a context, i.e. all desktop resources as well as other data structures created by the system, such as metadata in an RDF store and inverted index entries created for efficient textual search. We discuss different kinds of policies to specify protection for desktop data and metadata, and describe our access control system to express and execute these policies efficiently. Additionally, we describe the extension of an existing PIM system, Beagle++, with our approach, as well as our experiments, with convincing results on performance and scalability.

1 Introduction

In recent years, the amount of available digital information has increased considerably not only on the Web but also on personal computers. New innovative Personal Information Management (PIM) systems support users in organizing and managing their calendars, e-mails, address books, and other information on their desktop. PIM systems like Google Desktop [8], Beagle [2], Haystack [11], and Gnowsis [15], define semantic data as all content of the personal information space. Semantic data thus include the actual desktop resources and all additional data structures the PIM system creates, such as extracted metadata, including all machine generated information describing the resources and appropriate data and index structures supporting the functionality of the PIM system. A promising extension of PIM systems is to move from the pure desktop data management system towards the sharing of information among different personal spaces and users [14, 4]. However, these systems are doomed to fail if they do not incorporate mechanisms to deal with privacy issues and access control, specifying and

* In alphabetical order

checking when and which semantic data is provided to whom. Appropriate mechanisms must allow users to control the sharing of both the actual resources and the metadata about them, as in many cases revealing the existence of a resource or even parts of the metadata is considered to be sensitive.

In this paper we discuss how to use policy languages [17, 12, 7, 3] to provide users with appropriate functionality to describe access control policies for their shared semantic data. To avoid expensive evaluation at run-time often incurred by such systems, we present an access control system that optimizes run-time execution of these queries, significantly reducing the response time and computer load of the personal computers queried.

The rest of this paper is organized as follows. §2 presents a motivating example and the requirements of an access control for semantic data. §3 presents different kinds of policies and explains how they can be efficiently executed using the access control mechanism we suggest. §4 describes our prototype implementation, and §5 presents the experimental evaluation we performed using this prototype. §6 discusses related work and §7 presents our conclusions.

2 Semantic Data Sharing

Let us consider Alice, who is an employee in a company aware of the great benefits of information sharing among co-workers. In this company, instead of large centralized repositories of information, a PIM system with sharing capabilities[1]. such as BEAGLE^{++} [4] is provided. Each user of the system has a *semantic desktop*, with a set of filters and generators to extract metadata from desktop resources (i.e., emails, publications), an RDF store to maintain this metadata, and an inverted index to allow full-text search. A graphical illustration of semantic desktops and semantic data sharing in our scenario is shown in Fig. 1

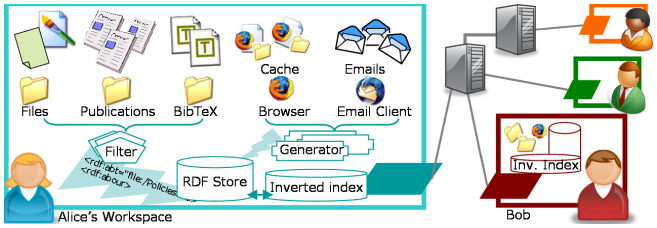


Fig. 1. An illustration of the semantic desktop architecture of the BEAGLE^{++} system and semantic data sharing between different desktops.

Alice has many resources on her desktop but she is not willing to automatically and unconditionally provide access to all her co-workers. Therefore, she creates policies to express the conditions under which she wants to share resources. For

[1] We assume that different desktops are connected with a P2P network such as Edutella [13], which does not require information to be shared among peers

example, for some project-related documents she states that only members of that project are granted full access to the resource, though the metadata about the title and authors are available also to non-members. Alice's co-workers are able to search for information on her desktop by sending queries to system. When Alice's semantic desktop receives these queries, her access control system ensures that all metadata and resources returned to her co-workers conform to the policies specified by Alice.

An access control mechanism for semantic data sharing between semantic desktops has some special requirements. One of the main requirements is typically to assume that everything is private by default, that is, nothing is shared unless explicitly stated otherwise. In our scenario, bad consequences of sharing sensitive data are more harmful than not sharing public information. Another important requirement is that the access control must consider two levels of protection, the metadata describing the resources and the resources themselves. Even if a user receives metadata about our resources —therefore knowing about its existence— that does not imply that the resource itself is publicly available. Also, it is required that normal employees (and not only qualified security administrators) must be able to personalize their policies, even if a default set of policies is provided. Furthermore, since the query execution will be performed at each employee's desktop, the access control mechanism must have a good performance.

3 Fine-Grained Access Control on the Semantic Desktop

This section describes the kind of policies considered in the paper as well as the main challenges addressed. It presents our solution which provides performant and fine-grained access control to information resources at run-time.

3.1 Specifying Policies

Policies specify the conditions that must be satisfied in order to grant access to semantic data. The policies described in our scenario can be classified into the following two main categories (examples are expressed with the PROTUNE policy language [3]):

(A) Resource Policies. These policies specify whether access to an actual resource (e.g., if the resource can be download) is granted or not. The conditions are described using the attributes found in the corresponding metadata. Some examples are listed in the following paragraphs.

Example 1. In our scenario, Alice gives access to any employee marked as co-author of a paper:

```
allow(access(file(Resource), Requester)) ⟵
    metadata(Resource, author, Requester).
```

(B) Metadata Policies. These policies state conditions under which different attributes from the metadata describing a specific resource can be disclosed.

Example 2. Another policy from our scenario states that only the subject field of e-mails not sent by her boss Tom are to be shared:

allow(access(metadata(subject, Resource), Requester)) ⟵
metadata(Resource, type, 'e-mail'), metadata('Tom', e-mail, TomAddress),
not metadata(Resource, from, TomAddress).

3.2 Query Processing and Policy Evaluation

When a request for information is received, the access control mechanism must evaluate all desktop policies and decide whether the semantic data to be delivered as search result can be disclosed. Remote requests (see Fig. 2) can be of two types: (a) *resource search* requests asking for metadata of resources relevant to a given query, and (b) *resource download* requests asking for retrieval of an actual resource. Resource search requests correspond to a user searching for resources matching a given query and therefore only return metadata about relevant results. A local inverted index is used in order to identify the resources relevant to the keywords of the query. For each resource, applicable policies have to be evaluated in order to decide whether a metadata field can be disclosed or not. The values of the set of granted fields for each resource are then retrieved from the metadata store and returned to the requester. For the resource download requests, the URI of a resource is given. The applicable policies for that resource are evaluated and the resource is sent back to the requester, if access is granted.

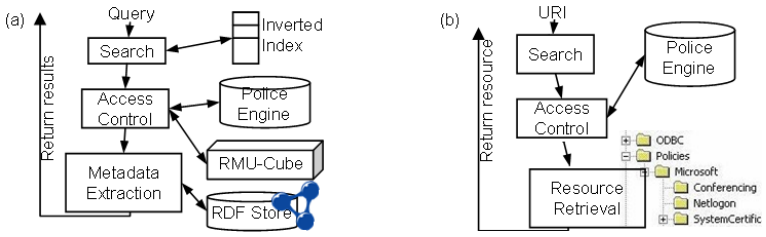


Fig. 2. The execution of (a) Resource Search and (b) Resource Download.

Obviously, for the resource search requests the system needs to evaluate applicable metadata policies whereas for the resource download requests, the system needs to evaluate applicable resource policies. Resource policies are applied only to a single given resource (the resource the remote user requested to download) and therefore the evaluation of the policies for that resource may imply an extra but acceptable performance cost. However, for metadata policies, this situation is quite different. For each relevant result returned by the inverted index, we need to check whether each metadata field can be disclosed or not. In order to make this decision, all applicable (potentially many) policies must be evaluated. Moreover,

each one of these policies may imply complex conditions as well as execution of some actions such as queries to the metadata store, e.g. to fetch status of the document. It is clear that metadata policy evaluation only at run-time is too expensive and not feasible for our scenario.

The following sections present some optimizations to dramatically decrease the time required to evaluate which metadata fields are available for each resource to a given requester.

3.3 Optimize Metadata Policy Execution

During pure run-time metadata policy evaluation, all policies must be evaluated for each metadata field of each relevant result returned by the inverted index. This evaluation can be quite time consuming since a policy may involve several actions such as requests to a metadata store. Assuming that the metadata describing the resources are rather static —usually these metadata do not change every minute— we can exploit the fact that also actions performed during the evaluation of the policies will not change much over time. We can therefore improve evaluation costs considerably by pre-compiling the results of the evaluation of the policies for the parameters resource, metadata field being evaluated and requester.

Our solution uses a three dimensional bitmap named RMU-Cube [2] (Fig. 3). The first dimension (vertical in our figure) represents the resources found in our workspace. The second dimension (horizontal) represents the different metadata attributes available for resources. The third dimension (depth) represents the set of users that may act as requesters. This list of users can either be updated manually or possibly automatically by a remote service offered by the company (in order to keep it up-to-date with employees joining or leaving). Each cell of the RMU-Cube represents the result of the evaluation of all policies for a specific resource, metadata attribute and requester. The cells may take two values: access granted (represented by a 1) or access denied (represented by a 0).

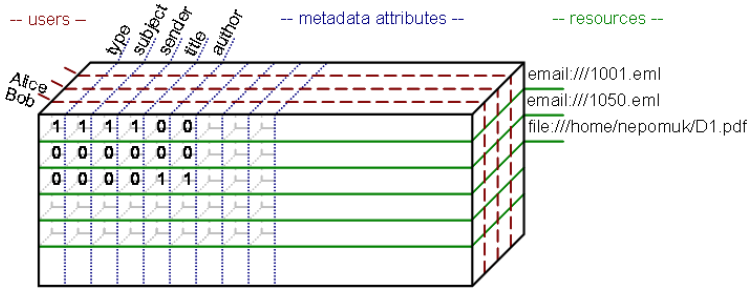


Fig. 3. The RMU-Cube represents the Metadata-Policies specified by Alice.

[2] RMU-Cube stands for Resource-Metadata-User Cube.

Example 3. Consider again Alice's semantic desktop. Among many others, it includes the following four resources: an e-mail *email:///1001.eml* which contains Tom in *cc*, another e-mail *email:///1050.eml* she sent to a colleague (without Tom in any field of the e-mail), and two documents *file:///home/nepomuk/finances.pdf* and *file:///home/nepomuk/D1.pdf* stored in the "/home/nepomuk" directory with status "Confidential" and "Final" respectively. According to these resources and the policies from §3.1 we can build the RMU-Cube depicted in Fig. 3. Metadata attributes that do not apply to a resource (e.g., cc in a normal document or author in an e-mail) are set to "access denied". The rest of the cells are set according to the pre-evaluation performed on the policies. Therefore, the subject of the e-mail *email:///1050.eml* can be shared with anyone, the title and author of *file:///home/nepomuk/D1.pdf* can be shared with anyone and members of the group Nepomuk may access all attributes of the documents *file:///home/nepomuk/finances.pdf* and *file:///home/nepomuk/D1.pdf*. The remaining metadata attributes are set to access denied.

Updating the RMU-Cube. An assumption when building the RMU-Cube is that resources, metadata attributes and users do not change so often that the updating mechanism of the RMU-Cube would overload the system. The removal of a resource, metadata attribute or user can be done quickly, since it only provokes the deletion of the corresponding plane. Additions or modifications are a bit more costly:

- Addition of a resource requires the creation of a new plane and evaluation of the applicable policies for each of its metadata attributes and potential requesters. Modification of existing resources does not imply an update in the cube unless some of its metadata is changed.
- Addition/modification of a metadata attribute requires the creation or re-evaluation of the corresponding plane according to applicable policies for each resource and potential requester.
- Addition of a user requires the creation of a new plane and evaluation of the applicable policies for each of the resources and its metadata attributes. As with resources, modification of a user (e.g., rename) does not imply an update in the RMU-Cube.

Assuming changes occurring as a result of normal user activity (e.g., editing of documents) and the further optimizations described in the next section, updates in our prototype can be performed without affecting the normal functioning of the desktop computer.

Finally, policies may change as well. If a policy is added, then all cells need to be re-evaluated. In case a policy language allowing only positive authorization policies is used (e.g., PROTUNE), only the cells set to 0 need to be re-evaluated (adding a new policy may only result on more permissions). If a policy is modified or removed, then we need to evaluate all cells of the cube. These updates are costly and therefore should be grouped so they are performed at the same time.

4 Prototype Implementation

We have developed the concepts presented in previous sections using BEAGLE^{++} as the reference PIM system. We reuse the following components from the architecture of BEAGLE^{++}: an RDF Store that contains the metadata describing the existing desktop resources (we use *Sesame 2.0*[3]), the desktop ontology providing the schema for the metadata and the inverted index for performing full-text search (we use a relational representation of the inverted index using *MySQL 5.0*[4]).

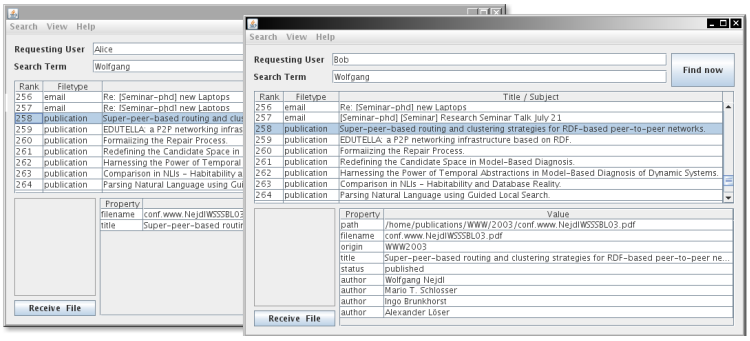


Fig. 4. Searching using the prototype implementation: Alice and Bob search using the same keyword, but Bob is able to access more metadata than Alice.

Integrating our access control mechanisms with the above components requires the following additional components:

Policy Engine: We use the PROTUNE policy engine in order to perform the evaluation of the policies. Its java API allowed us an easy integration with the rest of components. PROTUNE policy engine uses several threads for a more efficient policy evaluation and allows the execution of external actions such as queries to RDF stores, relational databases or LDAP servers.

RMU-Cube: This component is responsible for maintaining and accessing the pre-evaluated information of the policy engine in order to speed up the process of deciding which metadata attributes are accessible for a resource (see §3.3). We represented this RMU-Cube structure in a relational database using *MySQL 5.0*.

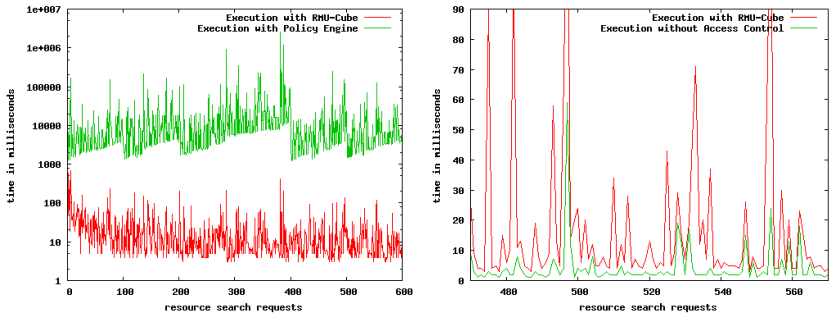
User Interface: In order to use the system, we implemented the simple user interface shown in Fig. 4. The interface receives the keywords and the requester name and returns the available metadata of matching resources. If the user wants to access a specific resource, the "Receive File" button allows the user to download it. Access control mechanisms take place on both situations in order to enforce the specified policies.

[3] http://www.openrdf.org/
[4] http://www.mysql.com/

5 Experimental Evaluation

We have performed several experiments in order to measure the impact of the concepts described in this paper. We used a large dataset including around 30 directories, 2200 publications from DBLP[5], and 2800 emails from the publicly available ENRON dataset[6]. Our dataset contained more than 5,000 resources and generated a total number of 72,974 triples in the RDF Store and 8,207 number of unique keywords in the inverted index.

In addition to this dataset, we implemented a similar scenario as the one presented in §2. We included a total number of 10 users and 20 policies protecting resources of the dataset. These desktop policies are similar to the ones presented in the examples of §3.1, granting access according to the attribute values of the metadata and the requesters/users defined by Alice[7].



(a) (b)

Fig. 5. (a) Comparison between time required to execute resource search requests using a policy engine and our RMU-Cube, and (b) total time needed for executing search requests without access control mechanisms or with our RMU-Cube.

We performed the experimental evaluation using a Pentium 4 computer with 1.5GB RAM. We executed a total of 600 resource search requests with different keywords which we randomly selected from the data in our inverted index. Fig. 5(a) shows the time needed for the execution of each one of the 600 requests with our RMU-Cube and with the run-time evaluation of the policy engine (we used a logarithmic scale for the time axis). As shown, the use of RMU-Cube dramatically reduces the time required to decide which metadata is allowed for each resource in comparison with the direct use of the police engine. Also, the integration of the RMU-Cube does not strongly impact the search mechanisms, since the difference between an execution without an access control and an execution with the RMU-Cube is only few milliseconds, as shown in Fig. 5(b)[8].

[5] `http://www.informatik.uni-trier.de/~ley/db/`

[6] `http://www.cs.cmu.edu/~enron/`

[7] The policies are available at `http://www.L3S.de/~ioannou/SDPolicies/`

[8] The peaks shown in the graph are produced by temporary overload while accessing the database in which the RMU-Cube is stored. However, average times show that

6 Related Work

In the last years we observed increasing popularity of systems for collaborative work and file sharing. The need for effective search within the increasing amount of information in this context pushed forward further development of search infrastructures for enterprise data management systems [10]. However, the sometimes private nature of such shared information makes it difficult to apply traditional document indexing schemes directly. The problem of applying traditional ranking algorithms to search through access-controlled collections is outlined in [5]. User access levels and access control have to be reflected in the index structures and/or retrieval algorithms as well as in ranking the search results.

In the literature, several solutions addressing the problem of privacy preserving of the data stored on public remote servers, which typically provide a basis for the community platforms, have been proposed. For example cryptographic techniques can enable users to store encrypted text files on a remote server and retrieve them using keyword search [9, 6, 16]. However, these solutions are not suitable for the collaborative multi-user environment. Alternatively, the data shared within a community can be stored locally by the user within an access-controlled collection. In this case efficient retrieval algorithms for search through access-controlled collections need to be provided to enable information sharing within the community. The authors of [1] address the problem of providing privacy-preserving search over distributed access-controlled content. Although this technique enables probabilistic provider selection it does not allow ranking of search results obtained from different document collections. Our semantically enriched community platform should allow providing unified view on the whole information set available to the user.

7 Conclusions and Future Work

Sharing desktop information requires scalable and effective access control mechanisms. In this paper, we have presented an approach that exploits the power of flexible and expressive policies and at the same time enforces them without impacting on the user's computer. Queries can be answered and information may be shared without perceivably increasing the response time of queries or overloading the personal desktop being queried. This approach is based on the pre-evaluation of the policies and its storage in a fast-accessible form (RMU-Cube), allowing for quick decisions for both the desktop resources and the metadata. Our experiments show how the use of an RMU-Cube dramatically reduces the computation and response time of enforcing access control on resources and metadata and how the integration of this mechanisms only provides a slightly higher response time than same queries without access control enforcement. We are currently optimizing our implementation and exploring and evaluating efficient techniques

the addition of the RMU-Cube supposes only some extra milliseconds in the process. Using e.g. an in-memory RMU-Cube representation would avoid such peaks.

for the update of this structure in order to face the evolution of desktop data and metadata. We are also integrating into a desktop agent which crawls the local files, extracts their metadata and index them in order to be shared.

References

1. Mayank Bawa, Roberto J. Bayardo Jr., and Rakesh Agrawal. Privacy-preserving indexing of documents on the network. In *VLDB*, pages 922–933, 2003.
2. Beagle search tool. `http://beagle-project.org/`.
3. Piero A. Bonatti and Daniel Olmedilla. Driving and monitoring provisional trust negotiation with metapolicies. In *6th IEEE POLICY*, pages 14–23, Stockholm, Sweden, June 2005. IEEE Computer Society.
4. Ingo Brunkhorst, Paul Alexandru Chirita, Stefania Costache, Ekaterini Ioannou Julien Gaugaz, Tereza Iofciu, Enrico Minack, Wolfgang Nejdl, and Raluca Paiu. The beagle++ toolbox: Towards an extendable desktop search architecture. In *Semantic Desktop Workshop 2006*, November 2006. Athens, GA, USA.
5. Stefan Büttcher and Charles L. A. Clarke. A security model for full-text file system search in multi-user environments. In *FAST*, 2005.
6. Yan-Cheng Chang and Michael Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In *ACNS*, pages 442–455, 2005.
7. Rita Gavriloaie, Wolfgang Nejdl, Daniel Olmedilla, Kent E. Seamons, and Marianne Winslett. No registration needed: How to use declarative policies and negotiation to access sensitive resources on the semantic web. In *1st European Semantic Web Symposium (ESWS 2004)*, volume 3053, pages 342–356, Heraklion, Crete, Greece, May 2004. Springer.
8. Google desktop. `http://desktop.google.com/`.
9. Hakan Hacigümüs, Balakrishna R. Iyer, Chen Li, and Sharad Mehrotra. Executing sql over encrypted data in the database-service-provider model. In *SIGMOD Conference*, pages 216–227, 2002.
10. David Hawking. Challenges in enterprise search. In *ADC*, pages 15–24, 2004.
11. Haystack project. `http://haystack.lcs.mit.edu/`.
12. Lalana Kagal, Timothy W. Finin, and Anupam Joshi. A policy language for a pervasive computing environment. In *4th IEEE POLICY, 4-6 June 2003, Lake Como, Italy*, pages 63–. IEEE Computer Society, 2003.
13. W. Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmér, and T. Risch. Edutella: a p2p networking infrastructure based on rdf. In *WWW*, pages 604–615, 2002.
14. Nepomuk: The social semantic desktop. `http://nepomuk.semanticdesktop.org/`.
15. Leo Sauermann, Gunnar Aastrand Grimnes, Malte Kiesel, Christiaan Fluit, Heiko Maus, Dominik Heim, Danish Nadeem, Benjamin Horak, and Andreas Dengel. Semantic desktop 2.0: The gnowsis experience. In *International Semantic Web Conference*, pages 887–900, 2006.
16. Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *IEEE Symposium on Security and Privacy*, pages 44–55, 2000.
17. A. Uszok, J. Bradshaw, R. Jeffers, N. Suri, P. Hayes, M. Breedy, L. Bunch, M. Johnson, S. Kulkarni, and J. Lott. KAoS policy and domain services: Toward a description-logic approach to policy representation, deconfliction, and enforcement. In *POLICY*, page 93, 2003.