Web information management with access control*

Serge Abiteboul
INRIA Saclay
& LSV-ENS Cachan
serge.abiteboul@inria.fr

Alban Galland
INRIA Saclay
& LSV-ENS Cachan
alban.galland@inria.fr

Neoklis Polyzotis
University of California
Santa Cruz
alkis@cs.ucsc.edu

ABSTRACT

We investigate the problem of sharing private information on the Web, where the information is hosted on different machines that may use different access control and distribution schemes. We introduce a *distributed knowledge-base model*, termed WebdamExchange, that comprises logical statements for specifying data, access control, distribution and knowledge about other peers. The statements can be communicated, replicated, queried, and updated, while keeping track of time and provenance. This unified base allows applications to reason declaratively about what data is accessible, where it resides, and how to retrieve it securely.

1. INTRODUCTION

With the Web, notably social networks and Web 2.0 applications, the sharing of information is generalizing. Users bring data to the network and are willing to share with others, but also wish to control what portions of the data can be viewed or updated by others. Users would also like to access and update information if desired and entitled to. This is the setting of the present paper, namely the specification and sharing of information with access control in a distributed environment. We wish to do so with a similar level of security as in centralized systems, but we also want to leverage and accommodate the wide variety of systems already available on the Web.

Several studies have investigated the problem of distributed information management with access control [8, 9, 10, 11]. Similar to some of the previous studies, our approach, which we term WebdamExchange, employs a distributed knowledge base as its foundation. However, the originality of WebdamExchange is that the knowledge base unifies all information relevant to data management with access rights. Specifically, the knowledge base contains logical statements and rules to encode:

1. data (as in databases),
2. access rights, and credentials (e.g., cryptographic keys pair or login/password),
3. localization information (where to find some particular data) as well as knowledge about other peers (e.g., replication of their data or trust in it),
4. rules describing the policy of each peer,
5. the provenance of the information using time and trace of communication, and
6. possibly other kinds of information that we will not consider here such as ontologies, ontology mappings, beliefs, trust, etc.

The approach allows reasoning holistically about pieces of data, from where they can be retrieved and who has some particular access right on them. Moreover, the "logic" of participants may be described declaratively using rules, which facilitates the development of distributed Web applications.

We illustrate these ideas with an example. Suppose that user *Alice* is organizing a rock-climbing outing in Fontainebleau, and wishes to put together an application for the event that she will share with the members of her rock-climbing group, *Roc14*. Part of the data she needs is the list of the group members, which is stored on Facebook. To promote the event, she also wants to use pictures from previous outings, which the group members store on public sites, such as Picasa or Flickr, private Web sites and in an untrusted DHT that group member *Bob* set up. Finally, some information she will need comes from public Web services, e.g., she might use Google maps to obtain location information for bouldering areas in Fontainebleau. Using existing technology, *Alice* will have to use several different tools and APIs in order to check what data is available and from which Web services, whether she has access to it and finally determine how to retrieve it. In contrast, the same task can be performed in WebdamExchange by issuing a declarative query that requests the needed data. WebdamExchange will process the query over the unified knowledge base, thus dealing under the covers with the thorny issues of distribution and access rights. Specifically, the knowledge base provides all information about obtaining (from Facebook) the list of group members, finding where each of them stores outing pictures and getting the data using proper credentials. Note that WebdamExchange has to perform this reasoning in an extremely heterogeneous setting, where systems, access controls and ontologies (data organization) may vary across members of the group.

Overall, our thesis is that with *all* this information managed in a distributed knowledge base, and with reasoning,

*This work has been partially funded by the European Research Council under the European Community's Seventh Framework Programme (FP7/2007-2013) / ERC grant Webdam, agreement 226513. http://webdam.inria.fr/
Fourteenth International Workshop on the Web and Databases (WebDB 2011) June 12, 2011 - Athens, Greece

we can automate the retrieval of information with access rights.

From a formal viewpoint, the system consists of a set of *peers*, each with its own database and its own logic. The database contains logical facts capturing information such as documents, access control, credentials, localization, but also replicas of other peers' information. The peer logic is expressed in datalog-style rules. We build on the WEBDAMLOG language [3] that supports the exchange of facts and rules between peers. We extend the language in a number of ways, notably by introducing the notion of *principal* (e.g., the group *Roc14*) which is common in security.

The holistic approach advanced by WEBDAMEXCHANGE brings several distinct advantages:

Large spectrum Because the model is general, it can capture very different scenarios ranging from centralized systems (such as central servers) to massively distributed systems, with peers ranging from fully trusted to totally untrusted, providing encrypted or clear information. Furthermore, it can capture scenarios combining the previous cases, which are the reality of today's Web, in arbitrarily rich ways.

Formal model Because the model is formally defined, we can prove (or disprove) desirable properties for a system described with our model, such as soundness (data is only acquired legitimately) and completeness (all legitimate data may be acquired). This is in the spirit of [1] that uses logic to describe access control protocols. Also, peers may perform automatic reasoning using the knowledge base to obtain information on data, localization and access control.

Quality control Because our model addresses provenance and time, we can better control the quality of data. This is in-line with recent works on data provenance, e.g., [5]. We view time and provenance not as gadgets but as essential components of a solution for properly supporting access control in a distributed setting. In particular, this will enable us to detect who is responsible for misuses of the system.

Given the limited space, our focus in this paper is on distribution and access control, two essential aspects of the scenarios targeted by WEBDAMEXCHANGE. We show how these aspects can be specified in the model, and briefly discuss how the model captures existing methods for managing distribution and access control. Of course, there are several other interesting questions, e.g., related to data integration or the use of ontologies, which we plan to investigate in our future work.

The paper is organized as follows. In Section 2, we briefly present the general WEBDAMEXCHANGE model. Section 3 deals with access control, and Section 4 with distribution. In Section 5, we highlight the main features of the system we are implementing. We conclude in Section 6 with perspectives on future work.

2. THE MODEL

We consider autonomous peers exchanging data using messages. For this, we use a formally defined datalog-style language recently introduced for distributed data management, namely WEBDAMLOG [3]. The language is very influenced by Active XML [2] and Dedalus [6].

The main novel feature of WEBDAMLOG is *delegation*, that is, the possibility of installing a rule at another peer. Specifically, WEBDAMLOG has two main concepts: facts, which capture both local tuples and messages between peers, and rules, which are datalog-like rules. As in deductive databases, the model distinguishes between extensional relations that are defined by a finite set of ground facts and intentional relations that are defined by rules. Moreover, the relation names are explicitly localized on a peer. For example, $r@p$ means the relation r of peer p. One can use variables for both relation names and peers, and they can appear as data in relations.

The semantics of the global system is defined based on local semantics and the exchange of messages and rules. Intuitively, a given peer chooses how to move to another state based on its local state (a set of local facts and messages received from other peers) and its program. A move consists in (1) consuming the local facts, (2) deriving new local facts, which defines the next state, (3) deriving nonlocal facts, i.e., messages sent to other peers, and (4) modifying their programs via *delegations*, i.e., the action of installing a rule at some other peer.

We illustrate the main notions of the model using an example. Consider the following two facts in WEBDAMLOG:

```
calendar@Alice-iPhone(rockclimbing, 06/12/2011, Fontaine-
    bleau, Alice-iPhone).
roc14members@Alice-iPhone(Bob, Bob-laptop).
```

The first one is a calendar entry that Alice entered from her iPhone. The second one captures the information that Bob is a member of the group and that he keeps his calendar in his laptop. The following rule is used to include rockclimbing entries from Alice's agenda into the agendas of other members of the group, and in particular into Bob's agenda:

```
calendar@$peer(rockclimbing, $date, $place, Alice-iPhone) :-
    calendar@Alice-iPhone(rockclimbing, $date, $place,
        Alice-iPhone),
    roc14members@Alice-iPhone($name, $peer)
```

A name such as Alice-iPhone or Bob-laptop has to be understood as a physical address in the network, e.g., a URL. It identifies a *peer* with storage and processing resources. Observe that variable $peer is valuated to a peer when the program is activated, and the new fact or rule is sent to the corresponding peer. In this particular case, this notably results in pushing to Bob's laptop the following new fact:

```
calendar@Bob-laptop(rockclimbing, 06/12/2011, Fontaine-
    bleau, Alice-iPhone).
```

Even more interestingly, the following rule on Bob's laptop:

```
confirm@$peer(rockclimbing, $date, $place,Bob) :-
    calendar@Bob-laptop(rockclimbing, $date, $place, $peer),
    checkAvailability@Bob-iPhone($date);
```

delegates to Bob's iPhone the rule:

```
confirm@$peer(rockclimbing,06/12/2011, Fontainebleau,Bob)
    :- checkAvailability@Bob-iPhone(06/12/2011);
```

So, Bob's iPhone can directly send a confirmation message to Alice if Bob is available that day.

The complete details of WEBDAMLOG can be found in [3]. Here, we introduce two extensions that are essential for WEBDAMEXCHANGE, namely virtual principals and semistructured data.

Virtual principal. The WEBDAMLOG language is tailored to physical peers such as Alice's iPhone or Bob's laptop, capturing data exchange between them. In WEBDAMEXCHANGE, we call *principal* an entity that owns data and has access right delegations on the data of other principals. A peer is such a *physical* principal. WEBDAMEXCHANGE also supports *virtual* principals, e.g., a user such as *Alice*, or a group such as *Roc14*. Contrary to a peer, a virtual principal has no storage or processing resources, relying on peers for that. The notion of virtual principal is primarily used to specify and manage access rights. Essential issues are who stores the data of a virtual principal, and who has read/write access to them. Typically, physical principals will store and process data for virtual principals. They may also temporarily create *avatars* of virtual principals. For instance, an avatar of Alice is created on her iPhone when she wants to access and update data from this device. A virtual principal may delegate its access rights to a physical principal who will manage its data.

Formally, the extension is as follows. Besides facts of the form $r@p(u_1, ..., u_n)$ where p is a peer, we have facts $r@q(u_1, ..., u_n)$ where q is a virtual principal. Such a fact is stored on a physical peer p as an external fact, i.e.,

$$external@p(r, q, u_1, ..., u_n).$$

where *external* is a reserved relation name. When a relation about a virtual principal is used in a rule, the peer "resolves" it (using rules) to replace it by the *external* relations.

Note that the basic WEBDAMLOG model is strongly typed. On the other hand, an *external* relation needs to store tuples of arbitrary arity. We next turn the model into a semistructured data model, which fixes this typing issue.

Semistructured data model. Another problem we have to face on the Web is that the data is naturally semistructured. For instance, a climbing area may be recorded in *Roc14* as follows, using the standard syntax of JSON:

climbingSite@Roc14:{"id":"&cuvier",
"Name":"Cuvier", "ClimbingSiteType":"Bouldering area",
"Circuit": [{"idref":"circuit@Roc14&cuvier-orange"},
{"idref":"circuit@Roc14&cuvier-blue"},
{"idref":"circuit@Roc14&cuvier-red"}]
"GoogleMapsURL":"..." }

This fact is representing the Cuvier bouldering area that includes 3 circuits, orange, blue and red. The number of circuits may depend on the size of the bouldering area. Brackets denote collections and ampersands denote references. Note that &*cuvier-orange* identifies a document within the domain of the *circuit* relation of *Roc14*.

To simplify data management, we extend WEBDAMLOG with the notions of a *document* and a *collection*. A *document* is a coherent, self-contained piece of data, modeled by an XML tree (We use the equivalent JSON syntax in the examples). For example, we may have the following document `picture1@Bob` containing a jpeg picture:

picture1@Bob = {"Name":"picture1", "Type":"jpg",
"byteStream":"..."}

A document corresponds to a relation that contains the different versions of the document at different points in time. A document update therefore corresponds to adding a new fact to the relation.

A *collection* consists of a set of independent references. Collections are also used for access control and localization (specifically, we employ collections of *access control* and *where statements* respectively, which we will define later). Collections are updated by adding or removing references. For example,

pictures@Roc14 += picture1@Bob

adds a reference to the previous document to the collection of pictures of Roc14.

The standard request to a document or a collection is to ask for its "current" version, i.e., the last version of the document or the consolidated list of references of the collection. These notions raise the issue of consistency that is particularly critical for collections of access rights. For instance, it is important to decide whether a peer requested an update to a document "before" or "after" it obtains the right to perform such an update. Distribution greatly complicates the issue, and the distributed systems literature provides a host of techniques for different consistency models. Clearly, these techniques can be considered in our setting.

Statements, instructions and external knowledge. In WEBDAMLOG, there is no difference between a fact and a message. More precisely, if a peer p derives a fact $r@p(u_1, ..., u_n)$, this is a fact to store; if p derives $r@q(u_1, ..., u_n)$, for $q \neq p$, this is a message to send to q. The message is automatically accepted by q as a new fact. However, in our current setting, we want to allow the peer to process the fact based on its own logic. In this fashion, q will see the message as a request to insert the fact, but it may decide to not actually insert it in its local knowledge base.

We next detail this important distinction between an insertion request and the (actual logical) statement that may result from it. The following *statement* may for instance be installed by Alice-iPhone:

Alice-iPhone *states* climbingSite@Roc14={"id":"&cuvier", ...} **requester Alice**;

In other words, Alice-iPhone created a fact of the relation climbingSite@Roc14 (and typically stored it in its database). It is important to understand who the participants in such a fact are: Alice-iPhone *performed* the statement; Roc14 *owns* this piece of data; Alice *requested* this update. This last information is used to trace the provenance of the fact. Such a complicated fact model is necessary because it is very typical on the Web to have a principal (Alice) who has the right to state a fact of another principal (Roc14) but has to rely on another principal (Alice's iPhone) to perform this task.

How did we get there? Typically, Alice made the following request:

Alice *requests* climbingSite@Roc14 = {"id":"&cuvier", ...} **to Alice-iPhone**;

to her iPhone. Another example of a request is

Bob-laptop *requests* get climbingSite@Roc14&cuvier **to Alice-iPhone**;

which is a request of Bob to Alice's iPhone for some data. If Alice's iPhone can prove that Bob is entitled to have this data, it can send it to him. So, a peer may want to exchange

statements and messages previously exchanged with other peers.

It is important to clarify that statements are local data, stored by the principal who performs the statements. Similarly, the instructions are stored by the principal who receives them. To be able to exchange data and trace any related communication, we introduce another class of messages that can contain statements or instructions, and that are used in general to transfer knowledge. For example, Alice-iPhone may answer Bob's request using:

Alice-iPhone *says* Alice-iPhone *states*
climbingSite@Roc14= {"id":"&cuvier", ...} requester Alice **to Bob-laptop**;

We also say that Alice-iPhone performed the communication and that Bob received it. This fact is stored by Bob-laptop. One may also want to exchange rules. For example:

Alice-iPhone *says* Alice-iPhone *states*
climbingSite@Roc14:{...} :- climbingSite@Alice:{...} requester Alice **to Bob-latop**;

means that Alice-iPhone installed in the latop of Bob a view machinery to copy Alice's data to Roc14.

The model allows capturing provenance information. In short, we can follow the transmission of pieces of information since each message records who sends it and who receives it. We will see in the following section that this information is authenticated, so one can check the full trace of provenance. When we want to trace the communication, the peers must accept only instructions and well-formed chains of external knowledge from outside. Formally, the performer of an external knowledge has to be the performer of the contained statement or the receiver of the contained external knowledge or instruction.

We should also mention that timestamps are also attached to messages. It is shown in [3] that timestamps have an important impact on the expressive power of WEBDAMLOG. We timestamp knowledge as follows:

Alice-iPhone *states* climbingSite@Roc14={"id":"&cuvier", ...} requester Alice **at 06/12/2011 14:00:00**

In WEBDAMEXCHANGE, the timestamps are local to the performer, here Alice-iPhone. There is no global clock.

To support the claim that the WEBDAMEXCHANGE approach allows handling a wide variety of situations encountered on the Web, we consider in more detail access control and distribution. In both cases, we introduce generic statements and briefly discuss how they may be supported in different Web contexts.

3. ACCESS CONTROL

In WEBDAMEXCHANGE, the access control granularity is at the level of the principal. A principal may get some particular access right to all the data of another principal. For example, Alice may be in the write access control list of the Roc14 group. She may then delegate this right to Alice-iPhone and Alice-laptop. In general, the access control policies will not be able to prevent malicious principals to misbehave. But if enough information is recorded (notably provenance), the system will be able to identify malicious peers.

In a standard way, access control in WEBDAMEXCHANGE is based on access control lists and credentials. We consider first the different access rights. The `own` right is used for authentication of performers and for granting/revoking access rights. The `own` right also transmits the access rights of a principal: if p is owner of q and q has a right r on q', then p has also the right r on q'. The `write` right grants the right to edit basic documents and collections. The `append` and `remove` rights grant the right to append and remove elements of a basic collection respectively. The `read` right grants the right to read basic documents and collections. The `readAcc` right grants the right to read the list of access control. The `readWhere` and `writeWhere` rights will be introduced in the section on distribution.

The first way to grant rights is to use an access control list. For example, the following statement specifies that Alice-iPhone grants the `write` right to Bob on Roc14:

Alice-iPhone *states* writer@Roc14+=Bob requester Alice

We also use the notion of *credentials*, which consist of two abstract dual notions, namely *hint* and *secret*. Concretely, hints and secrets will depend on the particular means of enforcing access control. Based on these abstract notions, the model will be general enough to match a wide range of scenarios found on the Web.

To illustrate these notions, we consider next a standard use of RSA-based cryptographic protocols. But it is important to keep in mind that the model is designed to support other situations as well. Indeed, we will mention some further. For this protocol, the credentials are a pair of asymmetric RSA cryptographic keys. Alice-iPhone may specify the creation of a pair of hint and secret for `write` right of Roc14 as follows:

Alice-iPhone *states* writeHint@Roc14{} requester Alice
Alice-iPhone *states* writeSecret@Roc14{} requester Alice

where *writeHint* and *writeSecret* are keywords of the language. This results in creating a public key to be used as the hint and a private one, the secret. The system uses such keys to control actions on data. The keys are used to authenticate and protect data. For authentication, the data is signed using the private key. Everyone can verify the signature with the public key. More precisely, a statement will carry the proof that the principal who performed the corresponding update was indeed entitled to do so. For instance, a statement about a document of Roc14 is signed with the `write` private key of Roc14, and carries as well provenance information that specifies who performed and requested the update. Symmetrically, one can use the credentials to protect the data and enforce read access control. For example,

climbingSite@Roc14{...} ***protected*** **for reader@Roc14**

means that the content of the fact has been encrypted using the `read` public key of Roc14. Continuing with the RSA cryptographic protocol, in this case, the content is *encrypted* with a fresh DES symmetric key, that is in turn encrypted with the `read` RSA public key of Roc14. So only someone with the private key will be able to decrypt the data.

A principal who can create statements to enforce access policies (access control and credentials) essentially has delegation power. We choose to authorize this action with the strongest access right of our model, the `own` right. So, for instance, a statement giving Bob `write` access to Roc14 has

to be authenticated with the `own` credential of the updated principal.

Authentication is also used to verify provenance for instruction and external knowledge. These messages are authenticated with the `own` secret of their performer. The receiver can check the identity of the performer and later prove that it indeed received the message. For example, let us recall the previous instruction and external knowledge:

Bob *requests* get climbingSite@Roc14&cuvier to Alice-iPhone;
Alice-iPhone *says* Alice-iPhone *states* climbingSite@Roc14=
{"id"="&cuvier", ...} requester Alice to Bob;

The instruction is signed with Bob's `own` secret while the external knowledge is signed with the `own` secret of Alice-iPhone. The internal statement is signed with the `write` key of Roc14. Before answering, Alice-iPhone has to check that Bob has indeed the right to access the relations of Roc14, by looking for the corresponding `read` access control statement. Alice-iPhone also stores the instruction, so that it can later prove Bob indeed requested the data.

We now discuss how to use credentials with totally different means of enforcing access control. More precisely, we consider the access control provided by standard Web sites. In WebdamExchange, an existing Web site is viewed as a peer identified by its URL. Some wrapping software is used to have the Web site behave as a WebdamExchange peer with typically degraded functionalities. For instance, the provenance information may not be recorded on the Web site. From an access control viewpoint, the authentication of an HTTP Web site may simply be based on trusting the DNS. (Recent hacks of DNS servers have shown limitations.) Access may be public, i.e., the hint is the URL and there is no secret. Access may be limited to principals who know of the URL, i.e., there is no hint and the secret is the URL. One may also consider more secure Web sites. For instance, the hint may correspond to the URL of the Web site together with a Web certificate, to provide more guarantees for authentication. To protect read/write access, the secret may also be based on an ssh private key or login/password, possibly relying on `htaccess` files.

Clearly, an interesting direction for future work is to investigate more complex access control schemes such as tripartite authentication of OpenID or Facebook (based on user, application and server) or majority voting for updates in P2P setting. But the main point is that the model already supports commonly found means of supporting access control.

4. DISTRIBUTION

In this section, we consider distribution. As we did for access control, we illustrate that a wide range of standard situations found on the Web, from very simple to very sophisticated, can be handled with the model.

Distribution is captured by a particular collection, namely *where*. Facts in this relation specify on which peers some particular data may be found. For example, to specify that the list of members of the group Roc14 may be found in the Facebook group of Roc14, one can use the fact:

where@Roc14:{"docRef":"member"} +=
http://facebook.com/Roc14

(The URI *http://facebook.com/Roc14* is understood by the "wrapper" of Facebook as a denotation for that Facebook group.) Localization collections are updated as other collections by appending or removing hosts. The language introduces new rights for controlling who can decide where such data may be kept and found, namely, `readwhere` and `writewhere` rights.

The primary use of localization statements is to enable the localization of data to a principal. For instance, Bob typically has some basic information about Alice that states where to look for information about her, perhaps a Web page where she put some basic information. Starting from that, Bob is then able to learn that she keeps her list of friends on Facebook, her pictures on Picasa with a backup on her laptop, her music on her TV box at home with a copy on Bob's laptop, etc. So, just with an entry point (i.e., a kind of extended vCard electronic business card) Bob can find all the information he needs from her (and he has access to).

Again, the main advantage of our approach is that the system can handle the heterogeneity of the Web. In the same manner that hint/secret enabled using data protected by different access control protocols, the logical form of localization information abstracts away various communication protocols.

To conclude this section, we consider another extremely interesting aspect of the use of localization, i.e., the possibility to use complex and dynamic localization schemes such as gossiping or DHT. Consider first gossiping. This is a case where search cannot be separated from the actual management of data. To answer a particular query using gossiping, it is important to choose properly who to ask: who is more likely to have the information, to be the specialist, to be trustworthy, etc. By making localization a component of reasoning for answering queries, we simplify the task.

Now consider the use of a DHT. It has been shown recently that a DHT can be modeled using distributed datalog [7]. WebdamLog could also be used. However, even if a standard DHT such as Pastry is used, one can naturally define the `where` predicate using standard DHT functions as in:

(where@Roc14:{"docRef":$r} += $p) :- relation@Roc14:{$r},
hash@DHT($r@Roc14, $p)

The rule specifies that $p is added to the list of hosts of the relation $r of Roc14 if hash@DHT($r@Roc14, $p) holds. The evaluation of the hash@DHT predicate is supported using a standard Web service of the DHT, i.e., the classical localization service of the DHT that takes a keyword as input and returns the address of a peer that has the desired information.

This illustrates the versatility of WebdamExchange to describe very different kinds of systems. So, a unique setting allows expressing very different security policies, very different distribution policies, as well as other essential aspects for distributed data management ignored in this paper, such as translating from the ontology of one peer to that of another.

5. SYSTEM

We are currently implementing the WebdamExchange system. A prototype that supported a large portion of the described functionality was demonstrated at ICDE [4]. The implementation is progressing towards the full support of all the WebdamExchange features described in this paper. In this section, we briefly mention aspects of the implementation.

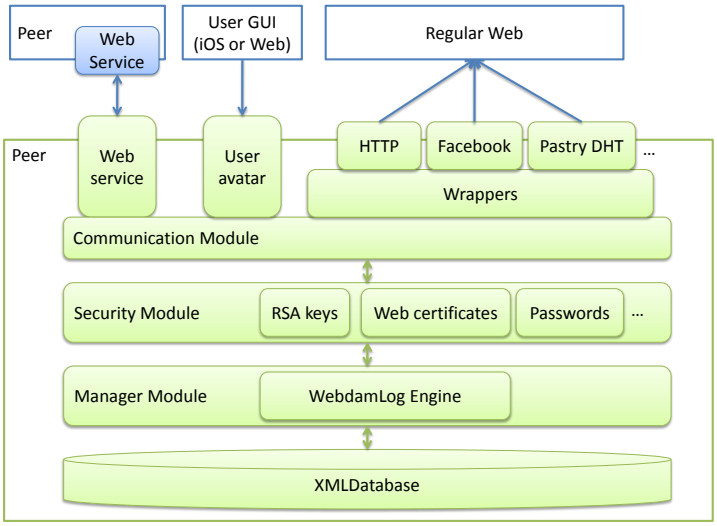

Figure 1: WebdamExchange Peer Architecture

A WebdamExchange (full) peer is designed as follows. It has a communication module, with a Web-service interface, to communicate with other peers. The messages go through a security module. It is the responsibility of that module to verify the origin of incoming data (e.g., by checking the RSA signatures) and decrypt incoming data. It is also its responsibility to authenticate outgoing data (e.g., by signing them) and protect them if requested (by encrypting their content). All the reasoning is performed in the engine that is responsible for localizing data and piloting the security module. For example, the policy of the system may be to trust everything that is signed by a particular "very-trusted"' user, even in absence of other credentials. More generally, the policy of the manager determines (among other responsibilities) whether an instruction is accepted or not and whether an outgoing fact is encrypted or not. The facts are stored in an XML database.

These modules are implemented in Java. So, in particular, policies are still "hardwired" and very limited in terms of reasoning capabilities. We intend to replace the manager by a WEBDAMLOG engine, so that data exchange will be guided by reasoning in the distributed knowledge base.

As mentioned in the paper, we want also to manage data that is in other popular Web systems. To enable such interactions, we have incorporated in the communication and security modules, plug-ins to support for the protocols of such systems (*wrappers* in mediator's terminology). We have been developing wrappers for standard Web sites, for secure ones, for Facebook and for the Pastry DHT. Also, the security module includes plug-ins for access by login and password and for authentication based on Web certificates.

Finally, we develop a special communication module for hosting avatars. It gives a principal the means to create a virtual peer communicating directly with the peer the principal is logged on. This module is used for the communications of a Web GUI with the rest of the system. We have also implemented an iOS GUI that can be used to interact with the system, e.g., from an iPhone.

6. CONCLUSION

We presented a model for Web data management with access control and distribution that captures a variety of protocols found on the Web. The main on-going task is the development of the WEBDAMLOG engine. An open issue is whether standard datalog optimization techniques are adapted to such a setting or whether there is a need for novel ones. Perhaps more critical, we need to understand how the rules are specified for a particular participant. A human participant is very likely to be reluctant to write rules. So, we have to define ways of selecting some from existing libraries and perhaps customizing them.

7. REFERENCES

[1] M. Abadi. Logic in Access Control (Tutorial Notes). In *Foundations of Security Analysis and Design V*, page 165, 2009.

[2] S. Abiteboul, O. Benjelloun, and T. Milo. The active xml project: an overview. *The VLDB Journal*, 17:1019–1040, 2008.

[3] S. Abiteboul, M. Bienvenu, A. Galland, and E. Antoine. A rule-based language for web data management. In *PODS*, 2011 (To appear).

[4] E. Antoine, A. Galland, K. Lyngbaek, A. Marian, and N. Polyzotis. Social networking on top of the webdamexchange system. In *International Conference on Data Engineering*, pages 1300–1303, 2011.

[5] P. Buneman and W.-C. Tan. Provenance in databases. In *SIGMOD*, pages 1171–1173, 2007.

[6] J. M. Hellerstein. The declarative imperative: experiences and conjectures in distributed logic. *SIGMOD*, 39(1):5–19, 2010.

[7] B. T. Loo, T. Condie, J. M. Hellerstein, P. Maniatis, T. Roscoe, and I. Stoica. Implementing declarative overlays. *SIGOPS*, 39(5):75–90, 2005.

[8] D. Mazieres, M. Kaminsky, M. Kaashoek, and E. Witchel. Separating key management from file system security. *SIGOPS*, 33(5):139, 1999.

[9] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, and J. Kubiatowicz. Pond: the OceanStore prototype. In *2nd USENIX Conference on File and Storage Technologies*, pages 1–14, 2003.

[10] A. Rowstron and P. Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. *SIGOPS*, 35(5):188–201, 2001.

[11] E. Wobber, M. Abadi, M. Burrows, and B. Lampson. Authentication in the Taos operating system. *TOCS*, 12(1):3–32, 1994.