"Share your data, keep your secrets."

Irini Fundulaki Daniel Lieuwen Arnaud Sahuguet
{fundulaki,lieuwen,sahuguet}@lucent.com
Bell Labs Research, NJ, USA

Guillaume Giraud Nicola Onose Nicola Pombourcq
Ecole Polytechnique, Palaiseau, France
{giraud,onose,pombourcq}@polytechnique.org

ABSTRACT

The next generation of services will require access to user profile information located on various networks. What's new is that this information needs not only to be integrated but also accessed in a selective fashion that respects the user's privacy concerns.

In this demo proposal we present a framework that performs the integration of distributed XML user profile components in a privacy-conscious manner. The key idea is to use a single language (XSquirrel) to deal with both data integration and access control and rely on language rewritings to statically process queries.

The demo will highlight the new features of our framework for some real life scenarios that involve XML user profile information.

1. MOTIVATION AND OVERVIEW

The next generation of services will not be restricted to the boundaries of a given network. This is called convergence! Land-line telephony, wireless telephony, instant messaging, Web, etc. now form a converged network where applications can be deployed. An example of such an application is the *Selective Reach Me (SRM)* which makes it possible for me (resp. other people) to reach people (resp. me) wherever they are (resp. I am). On the caller side, instead of calling a specific device, I call a person and I let the application figure out the best way to contact her. On the callee side, I want to specify who can contact me, on which device, when and for what purpose, etc. For instance, I may want to have the following policy: *people in my address book under category "business" can reach me on my office phone during working hours; family members can reach me on my cell phone anytime; others are directed to voice-mail.*

To make the above scenario possible, it becomes critical to provide a ubiquitous access to *user profile information* which (a) is distributed across networks and devices and (b) consists of static (e.g. identity information) and dynamic data (e.g. IM and wireless presence) and cannot be "warehoused".

Probably, even more importantly, is the critical need to control the way this information is accessed. The user is willing to disclose part of her profile information to certain users, but only if she can be sure that her information is not accessible by unauthorized parties.

The problems with the current architecture are that: (i) user profile information lives in network components with different protocols, data models, APIs. Consequently, applications need to deal with such heterogeneities which make the aggregation of information hard (if not impossible) and very expensive; (ii) access control is all-or-nothing and is spread across the various sources. For instance, access to my address book requires a password. If I want to share my address book, I need to give this password to other people or applications.

2. OUR APPROACH

Numerous industry initiatives like Microsoft Passport [12] and Liberty Alliance [8] have been started to address the issue of user profile data management. We describe here the GUP$^{\text{ster}}$ system which is motivated by the 3GPP Generic User Profile (GUP) effort [1], a telecom-based initiative that aims to aggregate user profile information relevant to network operators.

2.1 GUPster in a nutshell

The main idea behind GUP$^{\text{ster}}$ is to build an XML-based mediator that acts as a *centralized meta-data manager* to handle *highly distributed user profile XML data.* The mediator acts as the *single point of access* between data producers and data consumers. GUP$^{\text{ster}}$ aims to be the broker for user profile components which are (a) distributed across networks and (b) their distribution varies on a per user basis. Its role is two-fold: data integration and access control. This is the major difference with traditional mediator-based systems that only address the former.

2.2 Data Integration in GUP$^{\text{ster}}$

The backbone of the GUP$^{\text{ster}}$ mediator is an XML schema which describes user profile components and is defined by standard bodies.

The integration aspect is rather straightforward: our mediator needs to store mappings between user profile components (i.e. subtrees defined on the mediator schema) and data sources.

Queries are *user-centric*: a user (or an application running on behalf of her) (*requester*) initiates a request for a part of a user's profile data (*resource*), the latter expressed in terms of the mediator schema.

3. ONE LANGUAGE TO RULE THEM ALL

The key component behind GUPster is the XSquirrel language (see [6] for a full description of the language). Our requirements in GUP$^{\text{ster}}$are the following: we need a language that can express (1) views over XML documents for integration (i.e. mappings between sub-documents of the user profile document and remote sources), (2) views over XML documents to describe access control (i.e. association between sub-documents and boolean predicates) and (3) queries over these documents.

Note that, for both views and queries, we do not need/want to perform any restructuring on the document (there is no need for fancy joins). We only need to be able to specify what parts of a given document we need to keep. This is similar to the notion of *projected documents* introduced in [9].

The reason for "yet another language" for XML comes from the limitations of XPath and the high complexity of XQuery. XPath is very convenient when one or more subtrees of a given document need to be returned. However, using XPath, it is not very convenient to ask for parts of these subtrees. The second problem is that XPath returns nodes and not documents.

The problem with XQuery [3] is that it is too rich and powerful to reason about. Moreover, it requires one to deconstruct the document (creating bindings) and then reconstruct it, which means that we need to write rather complicated queries.

4. GUPSTER IN ACTION

We now provide an end-to-end example that shows how the access control rules and mappings are combined to answer an incoming query.

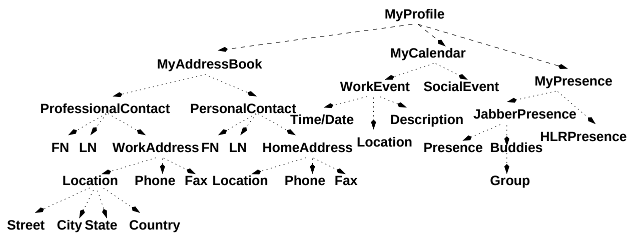

Figure 1: The GUP^{ster}schema

A simplified version of the schema we will use for the demo is presented in Fig. 1. In Fig. 2, we have gathered the mappings, access control rules and queries we use for this example.

Suppose that user Irini (or an application running on behalf of her) issues a query for Arnaud's address book, calendar information and mobile presence (from the HLR) during working hours. The resource part of the query is defined as the XSquirrel expression q_1 in Fig. 2. The syntax looks very similar to XPath with the added # operator to follow multiple paths at the same time. The access control rules R_1, R_2 and R_3 are relevant to the query (their requester part matches) and the condition is true. Moreover, we use the '\' axis instead of the / XPath axis to avoid any confusion.

GUP^{ster} first performs access control by trying to find the relevant access control rules. Then, for these rules it tries to rewrite the incoming query's resource part into an expression "compatible" with the resource part of the access control rules. From this point on, we use the terms query and access control rule to refer to their resource part.

For instance, query q_1 asks for the AddressBook. But access control rule R_1 only makes the PersonalContact under AddressBook accessible. For WorkEvent under MyCalendar requested by q_1, only the Description is accessible. The presence information (HLR under MyPresence) is not accessible (there exists no access control rule that specifies otherwise). Hence, by applying the relevant access control rules, query q_1 gets rewritten into query q_2 (see Fig. 2). Note that GUP^{ster} tries to find the "biggest" query permissible by the access control rules, as opposed to denying the query if it is not exactly permitted.

The next step is to perform data integration, i.e. mapping parts of the query to the various data sources that actually hold the data (by using the data mappings). From q_2, we can see that we need to access sources my.yahoo.com and my.netscape.com (mappings M_2 and M_3 respectively) to get PersonalContact. In order to get the calendar information we need to access source my.lucent.com (mapping M_1). After query rewriting, we have the following plan: query q_3 will be sent to source my.lucent.com while query q_4 will be sent to sources my.yahoo.com and my.netscape.com. Note that all of this has been done statically, without accessing the document itself.

In the final step, the various components obtained from the sources will then be merged to produce the final result returned to the requester.

5. OUR PROTOTYPE

M_1	*\MyProfile\(MyAddressBook\ProfessionalContact # MyCalendar\WorkEvent)*	$\rightarrow$	my.lucent.com
M_2	*\MyProfile\(MyAddressBook\PersonnalContact # MyCalendar\SocialEvent)*	$\rightarrow$	my.yahoo.com
M_3	*\MyProfile\MyAddressBook\PersonnalContact*	$\rightarrow$	my.netscape.com

Data Mappings for user *Arnaud*

Rule	Requester	Resource	Condition
R_1	Irini	*\MyProfile\MyAddressBook\PersonalContact*	
R_2	Irini	*\MyProfile\MyCalendar\WorkEvent\Description*	
R_3	Irini	*\MyProfile\MyPresence\MyJabberPresence*	9am $\leq t \leq$ 5pm

Access Control Rules for user *Arnaud*

q_1	*\MyProfile\(MyAddressBook # MyCalendar # MyPresence\HLR)*
q_2	*\MyProfile\(MyAddressBook\PersonalContact # MyCalendar\WorkEvent\Description)*
q_3	*\MyProfile\MyCalendar\WorkEvent\Description*
q_4	*\MyProfile\MyAddressBook\PersonalContact*

Queries

Figure 2: Mappings, access control rules and queries (as XSquirrel expressions)

Our current prototype is implemented in Java. It performs the rewriting for both data integration and access control, using some primitive operations for our XSquirrel language (intersection, union, and composition of trees). Mappings and access control rules are both stored in a relational engine.

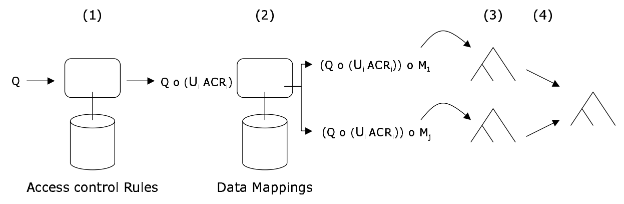

Figure 3: GUP$^{\text{ster}}$ Processing Flow

The processing flow is illustrated in Fig. 3. First we identify the relevant access control rules, and then we compose the query with their union. In order to evaluate the condition part of access control rules, our prototype packages access control decisions into XACML [11] decisions and invokes the Sun XACML [13] implementation (1). We then compose the rewritten query with relevant mappings to produce a query plan (2). Individual queries are sent to the various data sources, as SOAP messages (3). If needed, components retrieved from the various data sources are merged together (4).

We use the Axis framework[1] to package the various data sources and the GUP$^{\text{ster}}$ server itself as web services.

We currently support the following data sources, which we think are good representatives of different networks.
• address book and calendar information (from Microsoft Exchange, via WebDAV protocol);
• presence information (from Jabber server);
• personal information (from Lucent LDAP directory);
• presence and location (simulated from HLR data).

We have written wrappers to export the data of the above sources as XML data, compliant with the GUP$^{\text{ster}}$ schema.

6. OVERVIEW OF THE DEMO

For this demo, we will demonstrate:

- how to register user profile components (coming potentially from multiple sources) in the GUPster server.
- how to add/delete/modify access control policies (using the provisioning client presented in Fig. 4).

[1]http://ws.apache.org/axis

- how the meta information dictates the result of incoming queries, based on registered user profile components, access control policies, and request context (identity of the requester, other parameters such as time of the day).
- how GUPster can be used by applications and services via the SOAP interfaces.

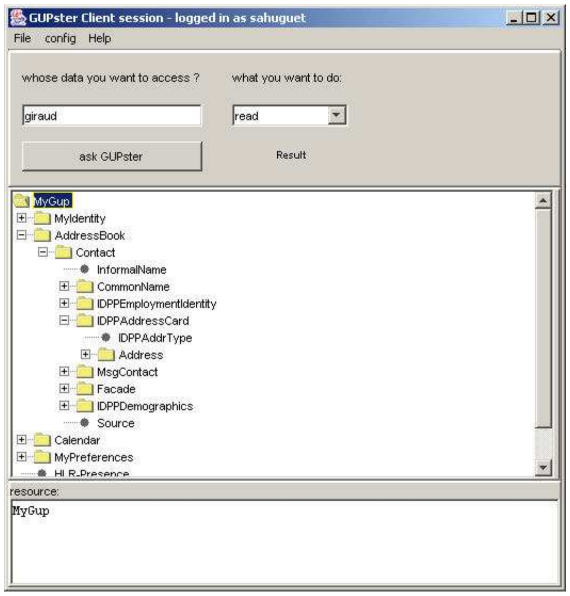

Figure 4: The GUP^{ster} rule provisioning client

More specifically, we will present 3 possible instances of GUP data consumers that we have built:

- personal web portal implemented on the server-side, including personal information, calendar, presence, locale (e.g. favorite locations, preferred language, currency). For instance, our personal portal can display weather forecast expressed in the preferred way (language, temperature units).
- personal web portal implemented on the client-side (making SOAP calls from Javascript using the Mozilla SOAP API).
- A device with limited capabilities (e.g. PDA or cell phone) using GUPster to synchronize some user profile information.

7. REFERENCES

[1] The Third Generation Partnership Project. http://www.3gpp.org.

[2] E. Bertino and E. Ferrari. Secure and Selective Dissemination of XML Documents. *ACM Transactions on Information and System Security*, 5(3):290–331, 2002.

[3] D. Chamberlin, D. Florescu, J. Robie, J. Simeon, and L. Stefanescu. XQuery: A Query Language for XML. http://www.w3.org/TR/xquery, February 2001.

[4] J. Clark and S. DeRose (eds.). XML Path Language (XPath) Version 1.0. W3C Recommendation, November 1999. http://www.w3c.org/TR/xpath.

[5] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati. A Fine-Grained Access Control System for XML Documents. *ACM Transactions on Information and System Security (TISSEC)*, 5(2):169–202, May 2002.

[6] I. Fundulaki and A. Sahuguet. Privacy Conscious User Profile Data Management with GUPster. Technical report, Bell Laboratories, Lucent Technologies, 2003.

[7] A. Gabillon and E. Bruno. Regulating Access to XML Documents. In *Proc. of the 15th Annual IFIP Working Conf. on Database and Application Security*, July 2001.

[8] Liberty Alliance Project. http://www.projectliberty.org.

[9] Amelie Marian and Jerome Simeon. Projecting XML Documents. In *Proceedings of the International Conference on Very Large Databases*, Berlin, Germany, September 2003.

[10] Makoto Murata, Akihiko Tozawa, and Michiharu Kudo. XML Access Control using Static Analysis. In *Proc. of the ACM Conf. on Computer and Communications Security*, Washington, DC, USA, October 2003. To appear.

[11] eXtensible Access Control Markup Language. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml.

[12] Microsoft Passport. http://www.passport.net.

[13] Sun's XACML Implementation. http://sunxacml.sourceforge.net/.