Enter Once, Share Everywhere: User Profile Management in Converged Networks

Arnaud Sahuguet Rick Hull Daniel Lieuwen Ming Xiong

Bell Laboratories/Lucent Technologies
700 Mountain Avenue, Murray Hill, NJ 07974, USA
{sahuguet,hull,lieuwen,xiong}@research.bell-labs.com

Abstract

Interoperation between network types (telephony, wireless, internet, etc.) is becoming increasingly feasible, leading to the so-called "converged network", and to a broad new family of end-user focused "converged services", that combine different kinds of network connectivity (voice, text-based instant and email messaging, real-time presence and location information, and access to numerous web-based services). A crucial but still largely unresolved issue is to provide easy – yet controlled – access and sharing of end-user profile data, including data about the user's devices, services, billing arrangements, address, calendar and preferences.

This paper surveys the kinds of profile data that are currently held in the various networks and where/how that data is stored, identifies key standards group activities working to enable profile data sharing, and proposes a data management framework (GUP^{ster}) that can be used across and within networks and organizations, to facilitate sharing of profile data for converged services. GUP^{ster} combines ideas from the federated architecture for data integration and the Napster approach for peer-to-peer sharing, in order to satisfy the high level requirements coming from standards bodies such as 3GPP GUP. The paper describes how the GUP^{ster} framework can be supported, and identifies key topics for future research.

Proceedings of the 2003 CIDR Conference

1 Introduction

The mobile telephony revolution has changed the behavior of millions. People can now reach and be reached anywhere, anytime. First limited to traditional voice networks (wireless and Public Switched Telephone Network, *a.k.a.*, PSTN), this revolution is now spreading across network boundaries to include the internet and other networks, giving rise to so-called *converged networks*. WAP was a first (perhaps unsuccessful?) way to bridge mobile telephony with the Web. The 2.5G and 3G wireless networks which followed already deliver email connectivity, instant messaging, and web access.

The convergence of networks has quite naturally given birth to a new breed of services – the so called *converged services* – which try to make the most of the network infrastructure in order to deliver value to end users and revenue to operators. These converged services revolve around providing information to end-users, providing connections between end-users with each other and/or with automated systems working on behalf of enterprises, and enabling end-users to invoke commercial, financial or other services outside of the network proper.

Providing these services entails the storage, sharing, and use by the converged network of a broad array of information about end-users, including, *e.g.,* the correspondence between users and their devices; awareness of presence, location, and availability of devices and their users; and awareness of billing models and plans (*e.g.,* pre-paid vs. post-paid). As network-hosted services become more complex, end-users will insist on highly personalized renditions of the services, which will entail the storage and access of large amounts of personal data (e.g., device specific information, usage preferences, address book, buddy lists, calendar, ...). This paper considers the data management technologies that will be needed to manage end-user profile information

in the context of converged services. In particular, the paper describes how and where profile data is currently stored in the various networks, surveys relevant industry standards groups, identifies key requirements for profile data management, and proposes a framework that fulfills these requirements.

The framework presented here, called GUP^{ster}, is based on three central ideas. First, a standardized schema for (most) user profile information will emerge from the activities of the 3GPP Generic User Profile (GUP) standards body [1]. Second, the peer-to-peer paradigm pioneered by Napster [13] will be adapted to allow profile data to continue to reside in widely distributed networks and locations, while information about *where* to find the profile data will be easily accessible, (*e.g.,* from a centralized meta-data manager analogous to the Napster server). Third, federated data management [8, 18, 25] techniques will be adapted to the specialized context of profile data management. Various other technologies used in the GUP^{ster} framework include technologies around XML, access control, data replication and synchronization.

The key contribution of this paper is identifying profile management for converged services as a crucial data management problem for the coming years, and providing a promising framework and structure for attacking this problem. While pointing the way towards a solution, the paper also raises a number of open research questions on how to flesh out the framework. This paper is focused primarily on the data management aspects arising in profile data for converged services. It is beyond the scope of this paper to provide detailed consideration of other technologies needed to support converged services, such as preference and policy management; workflow and collaborative systems; distributed systems; secure data transmission; standards for sharing information and control across networks and devices; new forms of call models and session management; and continued work on natural language interfaces.

This paper is organized as follows. Section 2 describes requirements for profile data management in support of converged services, by describing motivating examples and then listing the high-level requirements. Section 3 overviews the current state of the art in terms of data management for user profile in the various domains/network and gives an overview of the 3GPP GUP (Generic User Profile) initiative. Section 4 presents a high-level description of the GUP^{ster} framework, and Section 5 gives variations of the basic framework, more detail, and a discussion of how GUP^{ster} fulfills the requirements given in Section 2. We present some related work and open issues (Section 6) before we offer some future work and our conclusion in Section 7.

2 Converged Services: Examples & Requirements

This section overviews the requirements that a profile data management framework for converged services should satisfy. We present two illustrative examples, and then provide a listing of essential requirements.

2.1 Example 1: Roaming Profile

Consider a corporate employee Alice working for Lucent. She owns a SprintPCS cell phone that she uses for both business and personal matters. She stores her phone book and phone preferences (*e.g.,* ring tones, speed keys) on the phone. Alice's SprintPCS contract offers WAP capabilities, and she can store WAP bookmarks on her phone. When she travels abroad she also carries a GSM mobile phone operated by Vodafone. The SIM card she puts inside the phone contains her "European" phone book and phone preferences. Alice also owns a personal data assistant. Her personal address book and calendar information are synchronized with her Yahoo! account. Her corporate address book and calendar are securely hosted by Lucent.

As shown above, data is spread all over the place, in different networks. With existing technology, Alice's access to data is quite restricted, making it difficult or impossible for her to access her corporate calendar when she is traveling in Europe, share her address and phone book among SprintPCS, Vodafone and Yahoo!, etc. A framework for profile data management should enable easy and efficient access, sharing, update, and synchronization of this data.

2.2 Example 2: Selective reach-me

Mobile telephony makes it possible to contact people anytime, anywhere. This reach-me service can be customized in many ways. First, a callee can make (or program) the decision of accepting a call or not. Operators already offer basic call screening capabilities based on caller ID. Second, a callee can define the best way(s) to contact her.

In a converged network situation (see previous example), a user usually has more than one way to be reached. At work for instance, Alice can be reached (1) on her office phone (and voice mail if she is not in), (2) via email, (3) via instant messaging, (4) via VoIP (if she runs a soft phone like MSN messenger), etc. At home, she can be reached on her home phone. Potentially she can be reached anywhere on her cell phone, depending on the coverage. When she is near a WiFi hot-spot she can be

reached on her laptop via email, IM, and VoIP. Depending on Alice's location, the choice of the *best* communication medium varies: quality, price, nature of the communication, etc.

The selective reach-me service permits the network to optimally route a call (taken in its broadest sense) to reach Alice. To do so, the service needs to aggregate information for all the networks Alice is in contact with.

2.3 A Listing of the Essential Requirements

This subsection lists requirements that should be satisfied by a profile management framework for converged networks. To set the stage, we note that in the converged network, both profile data and services will be widely distributed. We now list the key requirements on a profile management framework. These are based on requirements identified by key standards bodies, and on experience at Bell Labs in providing support for existing and emerging converged services.

Common Data Model: Profile data should be accessible through a common data model, regardless of where or how it is physically stored or generated.

Data Richness: End-user profile information will become increasingly rich and varied as the services being supported become increasingly rich and varied. In particular, data may be deeply nested, involve complex integrity constraints, and for each end-user have different statistical characteristics.
Data Transformation: Data sources on the various networks have radically different structures and data models. Sharing data across these repositories, and also between network and terminal devices, will require efficient data transformation capabilities.

Data Placement: The placement of data will be dictated by end-user desires (the end-user may not trust some entities to hold profile data) and by optimization needs. In many cases, profile data will be stored redundantly (*e.g.,* telephone book may be stored in the end-user's phone, a "primary" copy held by an internet portal, and a cached copy held by a wireless service provider). Transformations may be used between redundant copies of data.

Data Integration: Many converged services will use profile data about a single end-user that is spread across multiple sources. Because the profile data may be distributed in different ways for each end-user, common tools should be provided to perform needed data integration. We expect data integration of profile data to be simpler than in the traditional setting, because profile-related queries do not typically require exotic joins: most of them are lookup queries like "retrieve presence information for Alice", "retrieve Alice's appointments for today", etc.

Data Reconciliation: Profile management must include mechanisms for reconciliation of slightly inconsistent data (*e.g.,* for synchronizing address book on phone with address book in network, or merging of address books from disparate places). End-users should be able to provision the policies used to reconcile profile data.

Data Synchronization: As indicated above, profile data will be cached, which implies the need for convenient synchronization mechanisms, and triggers to indicate when data has become stale.

Access to Meta-data: Because profile data will continue to be widely distributed, there must be mechanisms available for discovering where relevant profile data can be found.

Access Control: The end-user should be in control of what, when, how, and to whom profile data is shared. Mechanisms must be provided so that end-users can specify (possibly intricate) policies about access control (*e.g.,* presence data is revealed to co-workers only when the end-user is "at work").

Security: Data transmission should be secure. Authorization mechanisms need to be supported.

Data Provisioning: As the network-hosted profile data becomes richer and more voluminous for each end-user, it is increasingly important that end-users can provision (*i.e.,* insert, change, or delete) their profile data at will (self-provisioning). The provisioning should be accessible through a variety of interfaces, including large-screen web browser, handheld wireless device, and even voice. Provisioning interfaces should be automatically generated and should provide some guarantees (*e.g.,* constraint checking). Users should have the impression of "enter once, use everywhere", *i.e.,* the distribution and heterogeneity of the actual profile data should be transparent, except in connection with privacy of the data and trust in the organizations holding the data and/or meta-data.

Reliability: Telecommunication services must be responsive and simply cannot go down; wireline telephony is near real-time and 99.999% uptime is the norm. By merging telephony networks (PSTN and wireless) with other networks, users' expectation is to find the same quality of service with more features.

Scalability and Performance: Converged services have to be scalable to support millions of sub-

scribers. Therefore, data has to be partitioned and stored at different sites. Converged services have to be engineered based on the fact that the weakest link(s) will be part of the non-managed networks (*e.g.,* Internet). Many telecommunication services require real-time performance, *e.g.,*, in case of call delivery, response time of a transaction has to be within hundreds of milliseconds. Various optimization techniques will have to be used to overcome these limitations (*e.g.,* query optimization, caching, pre-fetching)

3 Profile Management in Converged Networks

This section gives an overview of the environment within which a profile management framework must reside. In 3.1 we discuss where profile data is found in the various networks today. In 3.2, we briefly discuss the activities of 3GPP GUP, the primary standards body currently working in the area of profile data. Perhaps the most important idea in this section is that we expect the 3GPP GUP standards group to be successful in creating and promoting a standardized schema that captures a substantial amount of end-user profile data; this standardized schema will become a key component of any future framework for profile management.

3.1 Overview of Profile Data in the Different Networks

PSTN networks: The Public Switched Telephone Network is the collection of interconnected public telephone networks, based on circuit-switching. To avoid a fully-meshed architecture, switches are used to connect together end-points. The role of the switch is to take care of the call control signaling, the media transmission and the logic. In current PSTN deployed architectures, the switch is a multi-purpose box (*e.g.,* Lucent 5ESS, the emerging softswitches) which encapsulates all these functionalities. User profile information is stored inside the switch itself, which makes it hard to access and extend. User profile information stored in switches depends on the nature of the services supported by the switch itself: call forwarding number, call barring numbers, caller id flag, 800-number resolution (when the user is a company), etc.

Wireless networks: Devices are connected to the network by a *Radio Access Network*. Subscriber information is stored by the *Home Location Register* (HLR), maintained by the subscriber home carrier. The HLR contains permanent subscriber information and the relevant temporary information of all subscribers permanently registered in the HLR. For each subscriber, its subscriber profile, location and authentication information are stored in HLR. Subscriber profile contains identity information, telephone numbers, and mobile user preferences related to services, such as call forwarding, barring, roaming, etc. A Visitor Location Register (VLR) maintains temporary subscriber information (snapshot of the master copy stored in the HLR) in order to handle requests from subscribers who are covered by the VLR. When a user moves from one cell (managed by a RAN) to another, a different VLR may be used.

Voice-over-IP (VoIP) Networks: VoIP networks – as the name implies – deliver voice using the Internet protocol (IP). The delivery of voice is initiated using a signaling protocol like SIP or H.323. Endpoints are IP phones or softphones (all software) which can talk the signaling protocol and process digitized voice using codecs. VoIP networks can be connected to other networks (mainly PSTN) using gateways that (1) perform the encoding/decoding of voice and (2) translate SIP addresses into phone numbers. Much of the "intelligence" (and therefore profile information) in VoIP networks is stored at the end-point. For instance, IP phones (soft or hard) usually store address book, call logs, forwarding preferences, etc. Some profile data is found in the network, however. In the case of SIP, network components consist almost exclusively of SIP registrars or SIP proxies. SIP registrars simply store a mapping between a SIP address (a VoIP phone number) and the corresponding IP address of the endpoint. SIP proxies are used for message routing and may store some user information.

Web: The Internet consists of machines connected via the IP protocols, some of them are servers, some of them are clients. We can distinguish between the public Internet (the Web) and the private Internet (the numerous intranets managed by corporations), usually isolated from the Internet by firewalls.

A broad variety of profile data is stored in the machines that form the internet, *e.g.,* bookmarks, address book, computer preferences, calendar information (iCal, vCal, Exchange), e-commerce profile data (shipping addresses, wish lists, buying patterns, etc.), employee information stored in directory servers, presence information (e.g. instant messaging client, connection to DHCP servers, etc.), web site log files, edge-router caching and routing policies, cross network info: ISP info about a user being connected or not and its IP address and calling phone number (in the case of a modem), etc.

Note that solutions to unify web-accessible profile data have been proposed. Netscape *roaming*

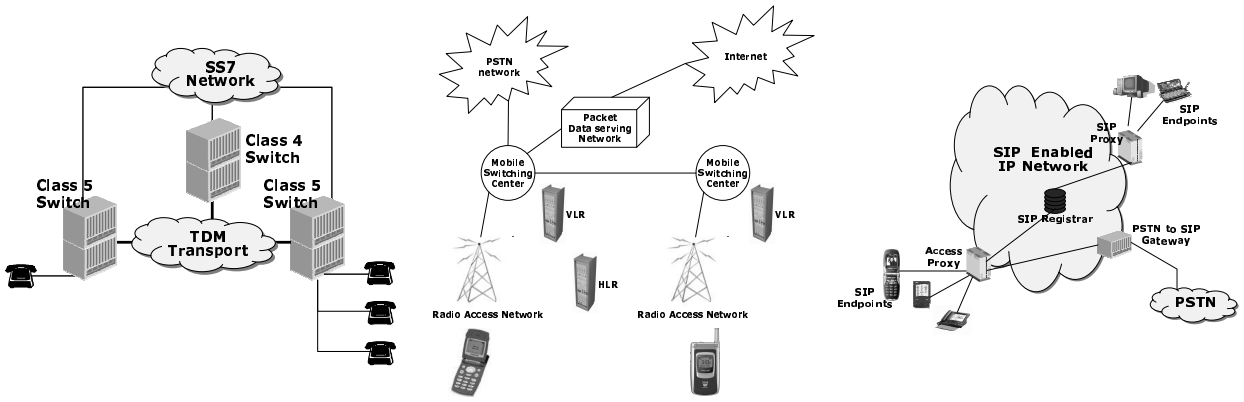

Figure 1: Converged Networks: PSTN (left), Wireless (center) and VoIP (right)

profiles are based on (1) a schema describing a limited set of user profile information and (2) a client/server protocol (LDAP or HTTP). Roaming profiles make it possible for a user to store her preferences (address book, bookmarks, cookies, browsing history) in a central server and have access to it from any client.

In this section we have seen that user profile information is spread across many machines and organizations on the various networks (see Figure 2).

Network	Locations of Profile Data
PSTN	Class 5 switches, billing systems
Wireless	HLR, VLR, MSC, billing systems
VoIP	end-user device, SIP registrar/proxy, AAA
Web	end-user device, ISP, portal, e-merchant, enterprise, edge-router, ...

Figure 2: Where profile data is stored

3.2 Overview of 3GPP GUP Goals

Several standards bodies and consortiums address the issue of profile data. Some view preference information as a kind of data that should be maintained alongside profile information. We focus here primarily on GUP. Other efforts will be presented as part of related work (Section 6).

3.2.1 3GPP Generic User Profile (GUP)

The 3rd Generation Partnership Project (3GPP) is a collaboration agreement that was established in December 1998. The scope of 3GPP is to produce Technical Specifications and Technical Reports for a 3rd Generation Mobile System based on evolved GSM core networks and the radio access technologies that they support. As part of its numerous efforts, 3GGP has started the Generic User Profile effort as a way to standardize the management of user profile information.

3.2.2 GUP vision

The GUP vision starts by looking at the current problems:

Great quantities of data, spread all over: As mentioned previously on this section, data is stored in many entities (*e.g.,* network components, IT information systems, user devices). Numerous and incompatible schemas and models are used to represent and store the data. Moreover, data is not often reused – requiring redundant storage, which leads to inconsistencies and wasteful re-entry.

Terminal management: A key focus of GUP is to make the management of end user terminals (provisioning and synchronization) easier. GUP has already identified SyncML [19] as the protocol for synchronization.

Customer care: End user problems must be able to be diagnosed and fixed via the network.

Multiple protocols: There is a need to support protocols for communication between network components, between network components and terminals, and between terminals (*e.g.,* phone $\leftrightarrow$ laptop).

Incompatibilities across domains: Various domains often use incompatible data models, schemas, and protocols. Requirements are also radically different (*e.g.,* real-time, reliability, security, etc.).

3.2.3 GUP requirements

Even though GUP is still in its early stage, the following requirements for GUP have been identified: harmonized query/update interface, common transport mechanism, common security protocols, and privacy. Also needed is a harmonized data description providing an extensible, common data model that is compatible with GUP information model. (The information model consider a user profile as a collection of profile components. A component is

used as a unit of storage and access control. Components are linked together by the identity they refer to.) Although not a formal requirement of GUP, it is very likely that the GUP group will adopt XML as the underlying format for the common data model and for data exchange. It should be clear that the requirements we have listed in Section 2 are a natural generalization of the GUP requirements to support services across the converged network, not just across the 3G wireless network. At present, the GUP group has not specified a reference architecture that can be used to provide simple and focused access to profile information.

4 GUP^{ster} in a nutshell

This section introduces the high-level GUP^{ster} vision, in its most basic form. The following section presents a variety of extensions and alternative formulations.

4.1 Napster + GUP = GUP^{ster}

To summarize in one sentence what GUP^{ster} is all about, we can say that GUP^{ster} is to user profile components what Napster was to music files. In Napster, users willing to share music files join the Napster community by registering their files on the Napster central server. The server stores meta-data about files and users. When a user wants to retrieve a given file, it sends a request to the Napster server and gets back list of peers (other users) who have registered this very file. The user can then ask for the file directly from the peers. In GUP^{ster}, data stores willing to share user profile components join the GUP^{ster} community by registering their components on the GUP^{ster} server. The server stores meta-data about data stores and components (*e.g.,* location, access control, etc.). When an application (*e.g.,* client application, data store, etc.) wants to retrieve a given component, it sends a request to GUP^{ster} and gets back a list of data-stores which have registered this very component. The application can then ask for the component directly.

4.2 Overview of the GUP^{ster} architecture

The GUP^{ster} architecture is presented in Figure **??** and consists of client applications, GUP-enabled data-stores and the GUP^{ster} server.

Client applications are applications which need to access user profile information for both query and update. Selective reach-me presented in Section 2.2 is a good example.

Data-stores are components which store and manage user profile information, such as HLRs, presence servers, portal sites, etc. Data stores must be GUP-enabled participate in the GUP community. Concretely, this means that an adapter is put on top of the data store to offer a GUP-compliant interface (protocol and data model)[1].

The GUP^{ster} server is the central repository of meta-data regarding user profile components. Among other things, it stores coverage (how the GUP schema is mapped onto existing data stores) and access control information. Note that "central repository" has to be understood from a logical point of view and may be implemented as a constellation of connected servers. In the simplified context of this section, we envision GUP^{ster} being supported in a manner similar to the UDDI registry [21], i.e. a family of mirrored servers hosted by a consortium of enterprises and freely available to all users.

4.3 GUP^{ster} in action

We assume that the we have some data stores willing to share user profile components and that these components support the GUP interface. (Figure 3 shows an example of GUP^{ster} integrating data from multiple data stores.) We now present a simplified scenario for GUP^{ster} in action. Later, we will discuss the access control aspect.

Like for Napster, the first step is for a data store to register to GUP^{ster} the components it is willing to share. For instance, Yahoo! will tell GUP^{ster} that it stores the address book of Arnaud and the address book and game scores of Rick. Sprint PCS will inform GUP^{ster} that it stores Arnaud's address book and game scores. Data stores can also unregister components. For each user, GUP^{ster} maintains the *coverage* of profile components by data stores. In the case of Arnaud, the coverage would look like [2]:

Coverage

```
□ /user[@id='arnaud']/address-book ↦
                    { gup.yahoo.com, gup.spcs.com }
□ /user[@id='arnaud']/presence     ↦
                    { gup.spcs.com }
□ /user[@id='arnaud']/buddy-list   ↦
                    { gup.spcs.com }
□ /user[@id='arnaud']/payment      ↦
                    { gup.citibank.com }
```

A coverage is a mapping between sub-trees of the GUP schema (expressed as XPath expressions) and data-stores. Note that a given profile component can be mapped to multiple data-stores.

When a client application wants to retrieve or update a profile component, it needs to send the

[1] As of this writing, the details of the interface have not been finalized yet. The data model will be XML-based and the protocol will probably be SOAP or HTTP.

[2] We assume that the GUP schema is defined with elements such as `user`, `address-book`, `presence`, `buddy-list`, etc.

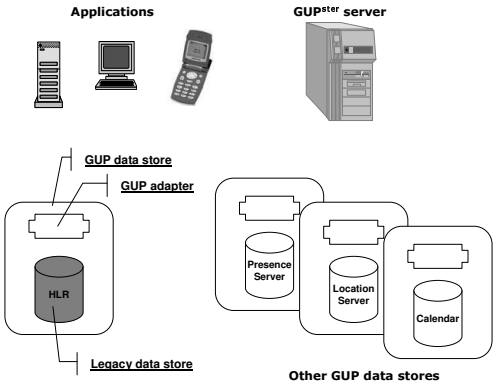

Figure 3: GUPster top-level architecture

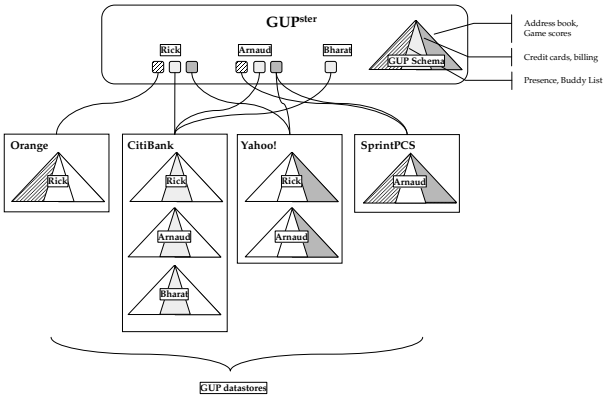

Figure 4: Schema coverage

request to GUPster. For instance, Arnaud wants to synchronize the address book on his cell-phone. The cell phone (through an application running on the cell phone) will send a request to GUPster. GUPster will rewrite the request and send it back to the client. In our example, GUPster will return to the client application something like:

Referral from GUPster

```
gup.yahoo.com/user[@id='arnaud']/address-book ||
gup.spcs.com/user[@id='arnaud']/address-book
```

where || has to be understood as a choice [16]. GUPster does not return any data, just a referral to be used by the client application. The client application will then use the referral (one of them, or both) to get the data directly from the GUP data stores.

4.4 Schema management

A key aspect of GUPster is to integrate profile components from a large number of data stores living in different networks. As we already noted, since the kind of queries we plan to support are not join-based, the "local as view" and "global as view" approaches essentially converge in this context. We assume that a global schema will be defined and maintained by a standard body (*e.g.,* 3GPP, W3C). The responsibility of the GUPster server and the various data stores is to make sure that the latest version of the schema is the one being used. Note that by design, the schema can be made more tolerant (or not) to evolutions (*e.g.,* using optional elements or attributes).

Here is a possible top-level schema for user profiles. The exact definition (and extensions) of the schema will be the responsibility of a standard body.

A possible schema for GUP

```
<MyProfile>
|
+-- <MySelf> (identity, address, etc.)
|
+-- <MyDevices (phone, PDA, laptop, etc.)
|
+-- <MyContacts> (<adress-book>, etc.)
|
+-- <MyLocations> (places where I may be reached)
|
+-- <MyEvents> (<calendar>, etc.)
|
+-- <MyWallet> (<banking-information>)
|
+-- <MyApplications>
    |
    +-- <Gaming>
```

4.5 Schema coverage

The schema coverage is a mapping between sub-trees of the GUP schema and user profile components available at the data-stores (see Figure 4). We assume that a profile component can always be defined as a sub-tree of the GUP schema. We exclude cases where, for instance, two profile components could not be defined as sub-trees but their join could be. The language we use to define coverage is a subset of XPath [24] with child- and attribute-axis only and limited predicates, in order to have a canonical way to navigate the tree.

For a given user, the GUPster server will store a list of mappings between a sub-tree of the GUP schema and one or more data stores. We have already presented some examples of coverage in the previous section.

Figure 5 has a more interesting example where Arnaud's address book is split between his corporate and personal address books. A request for `/user[@id='arnaud']/address-book` should return a referral to both data stores as well as a way to merge two XML fragments. We will discuss this issue in Section 6.

4.6 Access control

A critical aspect of the sharing of user profile components is privacy: users are willing to grant ac-

A more complex coverage

```
□ /user[@id='arnaud']/address-book/item[@type='personal'] ↦ { gup.yahoo.com }
□ /user[@id='arnaud']/address-book/item[@type='corporate'] ↦ { gup.lucent.com }
```

Figure 5: GUP^{ster} representation of address book split across two sites

cess to their profile information ("enter once, use everywhere"), provided they remain in control of who can access this information and when. Access control raises two issues: (i) how can the user specify her privacy shield, i.e. the set of access control rules/policies for her profile information; (ii) how are these policies being enforced.

4.6.1 Privacy shield

Conceptually the idea of a privacy shield is quite simple: should a given request be granted access to the data it asks for? The problem is to define what we mean by a request. The privacy is defined in terms of policies to be applied whenever some user profile is requested through GUP^{ster}.

As part of his privacy shield, for example, a corporate user may want to establish the following policies: *any co-worker can access my presence information during working-hours*; *my boss and my family can access my presence information at any time*; *my family can access my personal address book and calendar*.

In GUP^{ster}, a request consists of two facets: a context and a path. The path defines what components of the user profile are asked for. The context provides some information about the context of the request, i.e. identity of the requester (*e.g.,* third party application, end user, etc.), purpose of the request (*e.g.,* plain request, caching request, subscription-based request, etc.), etc. We envision the context to be an XML document as well, defined using a *request context schema*. The path can be defined as an XPath expression (returning nodes) over an instance of the GUP schema. The context can also be defined as an XPath expression (returning a boolean) over an instance of the request context schema.

We plan to reuse as much as we can from the emerging XACML [14] (XML Access Control Meta Language) standard. Unfortunately, the notion of request context in XACML is too limited (restricted to principals) to define a sufficiently rich privacy shield. As mentioned above, we will need to define an XML schema for the context information as well.

4.6.2 Policy infrastructure

Defining policies is one point; evaluating them and enforcing them is another. Issues to be solved are: who stores the policies, who evaluates them, and who enforces them

- policy repository: in charge of storing policies
- policy administration point: in charge of provisioning the rules (i.e. letting the user access and modify the rules that define her privacy shield) and other administrative tasks (*e.g.,* checking that the rules are valid)
- policy decision point: in charge of rendering a decision based on a rule set and a context. The decision point only returns a decision and has absolutely no side-effect on the environment
- policy enforcement point: in charge of asking for a decision and enforcing it (i.e. taking the required actions). Note that in some cases, the enforcement point uses a policy execution point.

Applied to our architecture, we envision the following distribution of roles (in the next section we will present variations where the roles can be assigned slightly differently). GUP^{ster} will be at the same time the administration point (allowing end-users to provision their policies), the policy repository (storing policies), the decision point (computing the decision) and the enforcement point (applying the decision, *i.e.,* sending or not a referral to the client). The data stores will be execution points. For 3rd party application, the policy infrastructure will be transparent: the application will send a request to GUP^{ster} and get back (or not) a referral.

5 GUP^{ster} Unleashed

In the previous section, we have presented the basic GUP^{ster} where all meta-data is stored in a centralized way and where the server only returns referrals. We now motivate and explore some variations of this architecture.

5.1 Architectural Variations

Maintaining and managing the profile meta-data is a central component of the GUP^{ster} paradigm. In the previous section, we described one architectural approach for how and where the meta-data might be managed, namely, a centralized meta-data manager. That approach was inspired directly from Napster, and assumes a UDDI-like universally available, mirrored meta-data store.

The primary reason to examine architectural alternatives is that two things are unpredictable at

present: (i) the degree of privacy about meta-data that end-users and enterprises will insist on, and (ii) the business model around meta-data management that will emerge.

With regards to privacy, consider for a moment Microsoft's Hailstorm initiative, which proposed a framework whereby end-user profile data could be stored at a central location by a single organization (MSN in this case) and could be shared on a selective basis with e-merchants at the "click of a button". This initiative was dropped because consumers were not willing to have a substantial amount of their profile controlled by MSN.

Many factors will be involved in the emergence of a business model (or models) that will arise to support a GUP^{ster}-style framework. A key question, of course, is how the organizations that provide meta-data management will be funded. Various alternatives exist, ranging from the end-user or the data requester paying for each use of the services explicitly (through a billing mechanism similar to current phone usage), to having the service bundled with other services (with the costs thereby "hidden" from the end-user), *e.g.,* as part of an internet portal or part of wireless phone service.

A simple variation of the centralized architecture is to assume that different end-users will want their meta-data managed by different organizations, perhaps their wireless or internet service provider, their bank, an internet portal, their employer, or even their home computer.

A further refinement is to allow the meta-data management for *each user* to be distributed. For example, a user might specify his primary meta-data manager to be their wireless service provider, but have meta-data about his credit card and other banking information be stored by a bank, and his meta-data about internet games be stored by an internet portal.

This suggests that there will not be a "one size fits all" implementation of GUP^{ster} nodes, but rather a handful of general implementations, and then several more narrowly targeted implementations.

5.2 Query Processing Variations

In the proposed architecture, GUP^{ster} is not doing any query processing only query rewriting. Its role is to combine coverage information and access control information to rewrite a client's request accordingly. GUP^{ster} does not return any data, only referrals. This means that a GUP^{ster} server is easier to implement and can serve more client concurrently.

But there might be some situations where having GUP^{ster} do some query processing would be useful. In the case of a client application with very limited capabilities (*e.g.,* a cell phone), the client may not be able to perform some required data transformation (like our address book example where the data is split among two stores and requires a merge). Offering a larger variety of *distributed query patterns* [16] like chaining, referral, recruiting (where the request is actually migrated to a different node) will be needed. GUP^{ster} should probably also offer some caching to make the access to user profile component faster. Another important dimension concerns subscriptions. In the current architecture, GUP^{ster} is a reactive (pull-based) not proactive (push-based) system. It is always possible to push-enable a pull-based system using polling, but this may not be very efficient. In our case, every polling request needs to be checked to enforce the end-user's privacy shield. Having the subscription handled by GUP^{ster} internally would save this extra work.

These variations can always be hard-coded into the various nodes of the system (the GUP^{ster} server and the various data stores). But it seems that a notion of *mobile query process* (as proposed in [16]) needs to be defined in order to capture the full range of query processing that may be needed to deploy such a distributed architecture.

5.3 How GUP^{ster} addresses the GUP requirements

We now revisit the requirements presented in Section 2.3 and show how they are being addressed by GUP^{ster}.

Data richness: Just like Napster has a built-in schema to "talk" about music files, GUP^{ster} maintains the schema which defines the structure of the various user profile components and their relationships. The structure of the GUP schema is defined using XML as the underlying data-model. Data stores willing to join the GUP^{ster} community need to export (virtually or physically) user profile components as XML, according to the GUP schema.

Data transformation: When the structure of the data hosted on the data store is not directly compatible with the GUP schema, we assume the existence of some wrappers/mediators in charge of transforming the data into the right structure. The transformation can be virtual or physical. In some extreme cases (*e.g.,* end user terminals with very limited query capabilities), we could imagine GUP^{ster} itself (or a component part of it) doing some further transformation if needed.

Data placement: We need to distinguish here between (i) placement decisions made by users (based on trust concerns) and (ii) placement decisions made by the infrastructure (based on performance concerns). For (i), the placement is dictated by the registration of components by data stores. For instance a user will instruct Yahoo! that it is responsible for storing its address book. Yahoo! will in turn register this component to GUP^{ster}: `/user[@id='Arnaud']/AddressBook => www.yahoo.com` For (ii), we can imagine that data stores operated by a given operator will be replicated, transparently for the rest of the network. Yahoo! for instance will store address book information in multiple servers (*e.g.,* `us-east.yahoo.com`, `us-west.yahoo.com`, `www.yahoo.co.uk`, etc.) and requests sent to `www.yahoo.com` will be routed to the closest Yahoo! store available. GUP^{ster} can also offer some caching services.

Data integration and Reconciliation: Data integration is provided by using adaptors to convert profile data from multiple repositories into a common format. Reconciliation can be handled by prioritizing sites or by some more sophisticated method when GUP^{ster} data is requested.

Data synchronization: The GUP working group has already agreed to use SyncML as the synchronization protocol. But this is only one aspect of the problem because SyncML [19] is only a transport protocol. Issues like synchronization semantics need to be addressed. GUP^{ster} does not provide specific solution for this yet.

Security and access control: GUP^{ster} does not plan to innovate in the field of security but will try to reuse the best existing solutions. An interesting issue is where the access control should be performed. We think that GUP^{ster} should be in charge of access control because it offers a single point of access. Having access control at the level of the data-stores would require keeping access control policies in sync. Here is a possible way to enforce access control. When an application sends a request to GUP^{ster} for a given component, GUP^{ster} checks whether or not access is granted. It rewrites the query accordingly (for instance only a subset of the information asked for can be returned) and signs it, including a timestamp. The application can send the rewritten and signed query to the corresponding data store(s). The store will check the time-stamp and the signature and eventually return the data. We assume that data store will only accept queries which have been signed by GUP^{ster}.

Reliability: Reliability will be achieved by having the logical single entry point be implemented by a constellation of GUP^{ster} servers.

Scalability and performance: As the logical single point of entry, before forwarding the queries, GUP^{ster} is able to filter out spurious ones (*e.g.,* queries which do not fit with the GUP schema, queries which do not satisfy the access control requirements). Since GUP^{ster} does not store any data, GUP^{ster} does not require a heavy duty database. GUP^{ster} needs to rewrite queries using coverage and access control information. By putting these function at the level of GUP^{ster}, we can keep the data stores as is (except for the GUP adapter on top of any data store) and expect very little overhead because of GUP^{ster}[3]. The use of multiple distributed query patterns (*e.g.,* chaining, referral, recruiting) will permit minimizing the transport cost of result information. The Napster analogy can give us some useful insights about scalability and performance. At its peak, Napster had more than 50m users with more than a few millions connected simultaneously.

6 Related Work

Profile Management efforts: There is a lot of on-going efforts in the area of user profile management. Both Sun (pushing for Liberty Alliance [11]) and Microsoft (pushing for Passport aka TrustBridge aka MyServices aka Hailstorm) are fighting over the next standard for network identity. Both initiatives are currently focusing more on the authentication aspect (single sign-on) than on the issue of data management. Liberty Alliance – as of this writing – has not proposed any data model and schema for user profile information. Microsoft's user profile only consists of the traditional user information stored by an ISP. A few years back, the DEN initiative [4] proposed a suite of schemas to describe network components and devices. Netscape roaming profiles are a direct extension of those. W3C Composite Capability/Preference Profiles (CC/PP) [23] initiative started in 1999 aims at creating *"a general, yet extensible framework for describing user preferences and device capabilities"*. The framework is based on RDF but does not seem to have received much support so far.

Personalization through policies: Personalization of converged services requires more than providing simple ways to input and access profile data; it requires the ability to specify and enforce a

[3]The GUP^{ster} architecture does not preclude network components from communicating with each other the way they used to. For instance, the presence component can retrieve location information from the HLR through GUP^{ster} or directly from the HLR.

broad range of preferences. Technology for storing and sharing preferences and their associated policies can build on the GUPster framework described in this paper. But considerable research is still needed in terms of how to provision and process preferences.

Standards bodies (e.g., Parlay policy management API [7], OPES [10]) and research activities are developing approaches to support the specification and enforcement of such elaborate preferences, typically through the use of rules engines. Various rules semantics might be used for different application areas, including systems with no chaining and systems that support production system style semantics [9].

XML integration: A lot of work has been done in the domain of XML data integration. Silkroute [6] enables the applications to specify virtual XML views over a relational database, and allows these views to be queried using XQuery. The IBM Xperanto [17] project and the Enosys XML Integration Platform [15] further allow the application to define and query XML views on data spanning multiple data sources.

As mentioned earlier (Section 2.3), query-oriented data integration of profile data is simpler because profile queries typically do not involve joins. As such, with respect to querying, these systems seem to offer more than GUP^{ster}. However, querying is just one aspect of data integration as supported in GUP^{ster}; apart from being queried, the data across the different data sources in GUP^{ster} needs to be updated (provisioned) in an integrated manner. None of the systems mentioned above handles integrated updates. Furthermore, GUP^{ster} also handles data placement (including caching) and reconciliation aspects (see Section 2.3) that are not addressed in any prior framework to the best of our knowledge.

The notion of coverage in GUP^{ster} requires some algorithms to decide query containment of XPath expressions (or subset of XPath), as studied in [5]. Some recent work on keys for XML [2] can also be applied to define coverage. New operators for merging XML components are also relevant (like Deep Union [3] or Merge [20]).

Security and access control: There has been a lot of work on security and access control in general. For XML, we already mention the emerging XACML proposal [14] and some of its limitations. XACML has a very restricted notion of request context. In GUP^{ster}, the quality of the privacy shield will depend on the context information the user can use to define her requirements. Moreover, XACML policies are expressed using some simple rules (no forward chaining) which are not suitable for complex access control policies. Offering an expressive framework with good enough performance is clearly a challenge. Another key issue is the provisioning interface that end users will use in order to express their preferences. An elegant solution for conditional access to XML (based on encryption) data has been presented in [12]. Some efficient algorithms for XML access control have been proposed in [27]. GUP^{ster} will be compliant with whatever standard gets accepted for network identity (*e.g.,* Liberty Alliance [11]).

7 Conclusion

This paper identifies a new and challenging direction in data management, namely to support easy (but controlled) access and sharing of profile data in support of converged services. These services often involve real-time, interactive communication between a user and other users and/or multiple network-hosted applications, and the associated profile data is typically spread across multiple networks and organizations. Additional aspects/requirements in this application area include the expectation that a single data schema will emerge for (most) profile data, that privacy and access controls are highly relevant, and the current lack of clarity on business models for managing profile data will be resolved. In addition to describing these directions in general terms, this paper describes the key kinds of profile data that need to be accessed in support of converged services, and also how and where that data is currently stored in the various networks.

The paper proposed a high-level framework for supporting access and sharing of profile data, which satisfies the requirements agreed upon to date by the 3GPP GUP standards body, and also follows the spirit of the Liberty Alliance consortium. This GUP^{ster} framework is fundamentally a combination of two things: the federated database paradigm and a peer-to-peer, Napster-inspired approach for organizing meta-data information. The fundamental technology challenges of GUP^{ster} are how to manage and control access to profile meta-data and the associated data, and how to provide efficient, scalable, and reliable support for sharing of the profile data.

While the GUP^{ster} proposal provides a sound basis for developing technologies for sharing profile data in support of converged services, it raises many challenges. We mention three of the most fundamental challenges here.

The core challenge is to continue work on meta-data management itself — what is the right conceptual model for meta-data (in the context of managing profile data), what are the efficient ways to store and access meta-data, what integrity constraints are relevant and how can they be enforced, is XPath sufficient for expressing the partitioning of meta-data and/or data that will be managed by disparate organizations, how should XACML [14] be adapted or extended, how should the Schema Adjunct Framework [22] be applied to capture these aspects, and what is a systematic framework for supporting the extension of the global profile schema (for both local and global extensions)?

A second challenge concerns efficient, scalable, reliable implementation. Although we presented some approaches in this direction, the area requires much more detailed consideration, including analysis of performance requirements, development of testbeds and benchmarks, and ultimately development of self-adapting implementations for data access, transformation, distribution, caching, etc.

The third core challenge involves data provenance [3], that is, the tracking of where data (and meta-data) have come from, and where they have been used. In e-commerce, when a user buys something, she gives her credit card number and the merchant gives a confirmation number for that specific purchase. The user trusts that the merchant won't use the credit card number beyond the purchases that the user authorizes. This illustrates just one example of the many kinds of tracking mechanisms that will be needed around access to profile data and meta-data. A special case of the data provenance challenge, in a different direction, concerns management of overlapping sets of profile data, *e.g.,* a user's personal and enterprise address books might be held by different organizations but hold overlapping data. What are systematic ways to support data reconciliation, to identify a single data source that holds all the data needed for a specific application, and to avoid distribution of data from one source that violates access controls given for another source?

Acknowledgements: the authors would like to thank Vinod Anupam, Bharat Kumar and Prasan Roy for comments and discussions about ideas presented in this paper.

References

[1] 3GPP. Generic User Profile, 2001. http://www.3gpp.org.

[2] P. Buneman, S. Davidson, W. Fan, C. Hara, and W. Tan. Keys for XML. In *WWW10*, May 2001.

[3] P. Buneman, A. Deutsch, and W. Tan. A deterministic model for semistructured data. In *Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats*, 1998. Available from http://db.cis.upenn.edu/DL/icdt.ps.gz.

[4] DEN, 2002. http://www.dmtf.org/spec/denh.html.

[5] A. Deutsch and V. Tannen. Containment and Integrity Constraints for XPath Fragments. In *KRDB 2001*, 2001.

[6] M. Fernandez, A. Morishima, D. Suciu, and W. Tan. Publishing Relational Data in XML: the SilkRoute Approach. *IEEE Data Engineering Bulletin*, 24(2), 2001.

[7] Parlay 3.0 Policy Management Specification, 2002. Available as the file Spec3_ParlayPolicyManagementDocuments1_0.ZIP at http://www.parlay.org/docs/.

[8] D. Heimbigner and D. McLeod. A federated architecture for information management. *TOIS*, 1985.

[9] R. Hull, B. Kumar, A. Sahuguet, and M. Xiong. Have It Your Way: Personalization of Network-Hosted Services. In *Advances in Databases, 19th British National Conference on Databases*, 2002.

[10] IETF. Open Pluggable Edge Services (OPES). http://www.ietf-opes.org.

[11] Liberty Alliance. http://www.projectliberty.org.

[12] G. Miklau and D. Suciu. Cryptographically Enforced Conditional Access for XML. In *Proceedings of WebDB*, 2002.

[13] Napster. http://www.napster.com.

[14] OASIS. eXtensible Access Control Markup Language. Available from http://www.oasis-open.org/committees/xacml/.

[15] Y. Papakonstantinou and V. Vassalos. Architecture and Implementation of an XQuery-based Information Integration Platform. *IEEE Data Engineering Bulletin*, 25(1):18–26, 2002.

[16] A. Sahuguet. *ubQL: a Distributed Query Language to Program Distributed Query Systems.* PhD thesis, University of Pennsylvania, Department of Computer and Information Science, 2002.

[17] J. Shanmugasundaram, J. Kiernan, E. Shekita, and C. Funderburk. Querying XML Views of Relational Data. In *VLDB*, 2001.

[18] A. Sheth. Federated database systems for managing distributed, heterogeneous, and autonomous databases. In *Proc. VLDB*, page 489, 1991.

[19] SyncML. http://www.syncml.org.

[20] K. Tufte and D. Maier. Aggregation and Accumulation of XML Data. *IEEE Data Engineering Bulletin*, 24(2):34–39, 2001.

[21] Universal Description, Discovery and Integration (UDDI) project. http://www.uddi.org.

[22] S. Vorthmann and L. Buck. Schema Adjunct Framework. Draft Specification 24 February 2000.

[23] W3C. Composite Capability/Preference Profiles (CC/PP). http://www.w3.org/TR/NOTE-CCPP/.

[24] W3C. XML Path Language (XPath). Available from http://www.w3.org/TR/xpath.

[25] G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 1992.

[26] J. Wilcox. Survey: Passport required–not appealing. CNET, April 2002. http://news.com.com/2100-1001-884730.html.

[27] T. Yu, D. Srivastava, L. Lakshmanan, and H. Jagadish. Compressed Accessibility Map: Efficient Access Control for XML. In *VLDB*, 2002.