Client –Side Profile Storage:
a means to put the user in control.

Stephanie Riché, Gavin Brebner HP Laboratories Grenoble.

Mickey Gittler Trusted E-Services Lab, HP Laboratories, Bristol.

09 November 2001

Abstract

On user-devices profile storage

As internet users, we provide personal information to a growing number of service providers with little or no control over its usage, and no means to properly track subsequent access of this information. Some companies have recently made announcements proposing to handle our personal information centrally, offering the possibility of a unified repository, but raising additional trust and privacy concerns. We propose an alternative to this trend by storing personal information on client devices, increasing the possibility of putting the user in control of his or her personal information.

A user can have multiple heterogeneous devices, so this generates a need for the distribution of profile data. Profile management capabilities are required that ensure consistency of replicated data, data accessibility with low latency, security and privacy. In our scheme we have chosen an approach based on a coherency protocol well adapted to handle data migration, and are extending this protocol to incorporate trust-related features.

1 Introduction

In today's approach to personalization, the technology is created and applied by those with a strong vested interest in tailoring commercial offerings to end users. Typically, personalization solutions are sold to e-commerce sites wishing to present the user with the combination of goods that will generate the most profit for the e-commerce provider. Although this may, in some cases, be aligned with the user's best interests, this is not the priority of the e-commerce provider. In particular, personal details of end users are a valuable commercial resource, and are often bought and sold without the users knowledge or consent. [1]

Another consequence of the service provider having control of the process is that a user typically will have multiple "profiles" across the Internet, with every site maintaining its own database of information. Personalization thus changes at every site, reflecting the information available, and can confuse a user who is unaware of what data (if any) the vendor possesses.

Recently, a number of profile-federation initiatives have been proposed. Microsoft has proposed Hailstorm (now known as My Services) incorporating some of its previous Passport system [2]. They propose to centralise (some) customer data in a paying service at Microsoft, and hence control a key part of the personalization experience, in the process siphoning off some revenue. Not surprisingly, many users and service providers have concerns with this approach. An alternative solution, recently announced, is the "Liberty Alliance" that aims to avoid a Microsoft monopoly through an open standard [3]. It remains to be seen what concrete proposals this project will make, but the scheme still appears to target server-side data ownership.

In all these cases, we see personalization technology moving away from real user ownership of profile data, and still being focussed primarily on targeting users for sales activity, rather than using the information for user benefit. We believe that an alternative exists, whereby personalization technologies are used to help the user, the primary objective of this alternative approach being to supply a better customer experience at the interface to the virtual world that includes the Internet and web services. We want to deliver technologies that allow devices to be better interfaces to the Internet. This objective implies a closer binding between user and device, with a greater need for trust between the two. The personalized device becomes a means to protect users, rather than help target them for business purposes. Given this, we work to the following:

1. A global vision where all devices in contact with an end user are aware of themselves, the user, and the local environment. This information is used to personalize interaction with the user, making interaction with the Internet simpler, easier, and more "human".
2. A view of privacy as a key enabler for this vision. The user has to trust the device / system to store personal data with privacy, and if the level of trust is insufficient, the clear, unencrypted data must not be left in the data store, or exchanged with a non-trusted 3rd party.

2 A user-centred architecture

A profile is used for the personalization of local and remote services. In addition, personal data can be used to customize man-machine interface [4]. We consider that data should be stored locally, permitting personalized actions even if the device is not connected to a network. In addition, local storage of data is fundamentally more private, as exposure of data can be limited to cases where a real user need is addressed. Examples of client side personalization are our previous work on VAL [5], and commercial products such as Belarc [6]. Client-side service execution is then made possible, thus reducing data exposure still further [7][8]. Also, it must not be forgotten that the processing power of client devices is still increasing, and that from processing power and scalability needs alone, a distributed client-side approach to personalization may make sense.

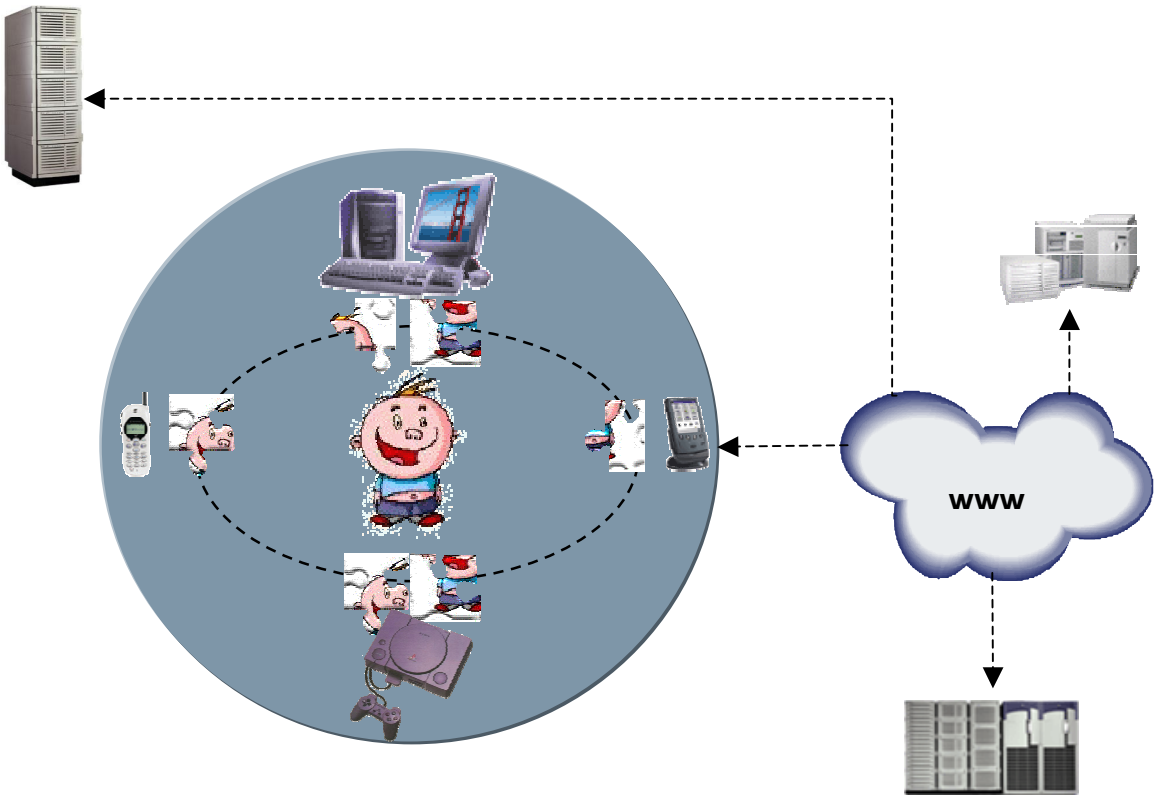

Figure 1: User centred architecture.

User devices form a trustworthy cluster. The profile is distributed and replicated on user devices. Each device can store part of the profile represented by jigsaw pieces. For instance, the mobile phone and the game console both have a copy of the same jigsaw piece. External system can access profile information but each exchange of data is under user control.

In consequence we propose a user-centric architecture. The user is central to the system and each user device becomes an interface to access services. Devices store profile information needed to personalize local and remote services, and collaborate to present a uniform and consistent view of a physically distributed and replicated system.

As illustrated in Figure 1, the user's heterogeneous devices form a trustworthy cluster. Each device can store part of the profile. Devices should be smart enough to handle the complexity of information management so that the user access and manage seamlessly his personal information. In our system, the profile is distributed, replicated and kept consistent on user devices. Thus the user can access any profile data from any device in a trusted way.

Device external to this cluster such as service provider servers can access part of the profile but this process is under the user control. We believe that client side architecture is intrinsically better adapted to user privacy through limiting the quantity of information that requires to be transmitted to third parties. In fact, exchange of information should begin with a phase of negotiation of what kind of information may be exchanged with a service provider. Our opinion is that exchange of personal information should be under user control, and that solutions should be provided to avoid intensely private information being sent to a service provider without the user's consent.

3 Profile characteristics and usage model

3.1 Profile definition

Whilst personalization is clearly defined as being the process of adapting services to an individual user, the word profile has multiple interpretations.

"Profile" is often used to refer to a set of user preferences / settings (e.g. the Unix Bourne shell *.profile* file). In the e-commerce world, it often refers to a set of user information (name, address, purchase preferences). In the telecom world, a similar set of characteristics makes up a profile. All these have the common theme of being the result of capturing certain user information and transforming them into a usable form. As rich user interaction requires implicit situational information [9] to increase ease of use and comprehensibility of user requests, we propose to extend this theme and to generalise the concept of a profile to refer to the digitised data on the user, taken from the surrounding context[10]. This profile may include any and all data that may prove useful in adapting a system to the user, thus personal details and preferences, devices used and their configuration, location data, transaction data, presence of other people and objects, and so on.

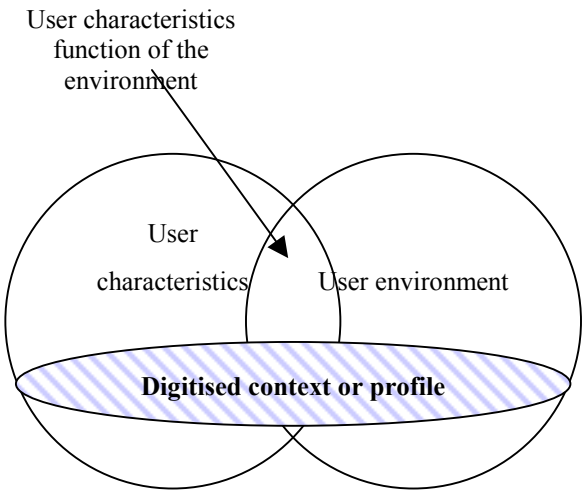

Figure 2: Profile or digitised context.

The disk labelled "user characteristics" and the disk labelled "user environment" represent abstractions of these notions in the physical world. A part of those characteristics are captured through devices and digitalized, forming the digitised context-aware profile.

3.2 Profile usage model

As noted above, we consider that each user will have multiple client devices, and that it is important for the same data to be available from any device; this means we need to consider how to distribute a profile store across multiple, intermittently connected, platforms. An important aspect of any such system is the provision for privacy. Privacy involves understanding of user privacy preferences expressed e.g. via policies such as P3P [11], and the execution of tasks in such a manner as to respect these policies.

We have decided to implement a shared profile store structure to obtain experience in how to build and access such distributed structures. We are equally interested in how different approaches behave in what we consider typical usage cases.

3.3 Identified Usage models

We consider several models of profile store usage. Not all are equally likely to be seen in real cases, and each present different demands to the profile store. We have attempted to characterize the different cases using two dimensions:

- **Grouped/Scattered Data:** Grouped data implies that the data used by an application forms part of a "clump" of similar data elements – e.g. all data can be found within some small sub-set of a profile. The alternative is for data to be scattered across the profile.
- **Migratory/Parallel:** Migratory data is data that "follows" the user from machine to machine. At any one time, the user is active on only one machine, and a change of machine is the event that may trigger the communication of any (outstanding) update of the data to another device. The alternative is for multiple simultaneous accesses to be made to data, causing frequent updates to be communicated between devices. We consider the simple case of a data item that is only ever used on one machine to be a trivial case of migratory data.

Some examples of the different cases are given below.

Case 1: Grouped, migratory – e.g. Radio Station Data.

This use case is illustrated by a demo we have constructed where the current radio "channel" is stored in the profile store. As the user goes from machine to machine, the radio channel setting follows and is consistent across many different machines. We thus have moderately static data (e.g. I don't change radio station all that often), used largely by a single application (radio player), well grouped with other radio data (if any), and used by large number of machines.

This use case is also interesting in that the data is used sequentially from a number of different machines. We see this as characteristic of a wide range of applications that are directly linked to user activity; while the user stays using a single machine, the data will not be accessed by any other machine, but when the user moves to another device, the (updated) data will need to follow.

Case 2: multiple-use, grouped data - e.g. Address data.

An example of this use case is that of the user's physical or email address. This data will be widely used, both across multiple machine and applications. The updates will generally be migratory (change once, not everywhere at once).

Case 3: Distributed, multi-application/machine data - e.g. CPU resources.

We can imagine a set of data relating to the CPU power available to the user. This is likely to be highly distributed in that device CPU power is probably grouped with other device-specific data, rather than grouped in a single sub-tree of the profile. The data will be used by many different applications and machines.

Updates to this data will come from changes in the environment (new machine being added, upgrade), and will typically be changed from only one machine. The data can therefore be grouped in with migratory data, in that simultaneous updates from more than one machine are unlikely.

3.4 Target Area

From our ruminations on usage models, we notice that most of the data we are considering currently follow a migratory pattern: data are accessed by applications residing on different machines in a predominantly sequential manner. Therefore, we have decided to focus primarily on migratory data for our initial research. Cases where parallel data access occur in large numbers are more likely to be experienced where the user possesses agents or other "do-it-for-me" systems that act on the user's behalf; such systems are currently much rarer than user-triggered examples. We will avoid making any decision that precludes the use of parallel data, but will not optimise the system for this case.

Hence, we are working on the management of migratory data by developing a research vehicle to experiment and get knowledge on how to store migratory profile data on user devices.

4 System overview

The system design presented below reflects the two main requirements on which we have focused:

1. A unique profile per user, that keeps up to date with user changes on any machine identified as belonging to that user.
2. Protection for user privacy.

As outlined in the introduction, we propose a client rather than server side solution to profiling, as we believe this allows personalization of local services in non-connected environment, and, just as importantly, enables solutions that improve user privacy, and increases user visibility of personal data usage.

4.1 Aims and principles

Our aim is to provide reliable profile data availability in a distributed environment composed of heterogeneous devices belonging to a single identifiable user. Initial work makes the simplifying assumption of one user, multiple machines, with each machine being owned by only one person. The set of devices form a trustworthy environment, raising additional issues such as how to control access to data annotated with various levels of trust.

The first model that comes to mind when you wish to share data between several devices is the client/server paradigm[1]. The server orders and processes access requests coming from various clients. However, for efficiency and availability a local presence of data is required, so it is usual to add a cache on the client. Caching is a form of replication, which is key to providing performance, high availability and fault tolerance in distributed system[12]. Performance is closely linked to the availability of data on the processing host, availability is relative to the number of copies of given data, and fault resilience is function of the level of distribution. In our approach, the role of server is distributed among devices, that is to say each device has a role of client and server enhancing fault tolerance potential.

Managing distributed data implies the definition of a naming scheme, that will allow invocation of the data from any device, the mapping of data names to communication addresses, and a coherence model that defines the behaviour of the whole system for managing concurrent access to shared data. Indeed, one of the most complex and important aspects when designing a distributed storage system is to resolve conflicting accesses to data.

4.2 Synchronization or shared memory?

Personal information management system approaches make use of a synchronization paradigm, like TrueSync[13]. This approach has the advantage of simplifying the management of communication; communication takes place relatively rarely and is explicitly triggered by a user event (e.g. putting a PDA in its cradle), and a failure of communication is not a major problem for the system (data is left unsynchronised).

However, the synchronized approach has a number of significant drawbacks:

- **Conflict resolution**. In a generic profile (as opposed to a limited data type system such as a contact database or calendar) generic conflict resolution has to be addressed. Reconciliation functions require an understanding of the data being synchronized; thus the profile store, and not just the applications, must be capable of understanding data types. We would greatly prefer a solution where the profile store

[1] Here, server and client are logical entities: A client requests a service or information to a server. Client/Server is primarily a relationship between processes running on different machines. The server is a provider of service or information, the client is a consumer of services. A server can be any type of device.

acts as a dumb storage mechanism, and does not have to consider the semantics of the data.

- **Weak coupling of devices**. Synchronization is a fairly weak coupling between copies of data. Data modification will be discovered after the changes, and not before. We want a solution that offers better visibility of the state of data copies. Any system using synchronization cannot rely on data being always valid, and this reduces the effectiveness of the end user application.

For these reasons, we have chosen to investigate an approach based on a shared memory paradigm. We thus are faced with an alternate set of problems, but consider we may obtain significant benefits from this less orthodox model.

4.3 The choice of a consistency model

One of our main challenges is to build technology to provide a consistent profile; i.e. that profile distribution is not visible to applications, and that the system behaves as if all data were stored on a unique locally held storage resource.

This challenge has been addressed in multi-processor systems to provide to the developer a programming model as close as possible to the one of single processor systems, yet with greater performance. Similar issues rose in other distributed systems supporting replication, namely distributed databases[14], distributed file systems -like CODA[15], NFS[16]- or groupware.

Three memory models encompass the set of well-known memory systems for the coherence problem.

- A memory is *strictly coherent* if the value returned by a read operation is the value written by the most recent write operation to the same object. Systems that realize this model offer the straightforward memory representation found in uni-processor systems at the cost of performance, as each participant of such a system needs to be aware of , and wait for, preceding write accesses, before it may access the same data.
- Munin [17]proposed a relaxed memory model based on the usage model of shared memory systems[2]. The second coherence model proposed by Munin[20] is defined as follows: Memory is *loosely coherent* if the value returned by a read operation is the value written by an update operation to the same object that could have immediately preceded the read operation in some legal schedule of the threads in execution. This approach offers better performance than strictly coherent systems,

[2] Indeed, in multiprocessor systems it has been proven that strict coherence is not necessary for correct execution of concurrent programs. Correctness of execution depends on the expected behaviour of the system[18], which is why the research effort has been mainly on the ordering of events by elaborating consistency models. A coherence protocol defines what value can be returned by a read operation and a consistency model adds a notion of event ordering by determining when a written value will be returned by a read operation. Dubois illustrates the close relation between those two models[19].

but provides a fairly complex memory model and raises the issue of chaotic access[3] as underlined by David Mosberger in his paper discussing memory consistency model [18].

- The third class of memory model mostly used in distributed systems, such as file systems or databases, is *eventual coherence* based on optimistic coherence protocols, such as epidemic algorithms used in Bayou[21] and reconciliation mechanisms. Memory is eventually coherent if updates are propagated such that eventually every copy has the latest version. Eventual coherence involves the design of reconciliation mechanisms as several copies of an object evolve independently from one another, resulting in possible conflict. Reconciliation mechanisms are a function of the semantics of the data to reconcile, and this approach is little better than the synchronization approach.

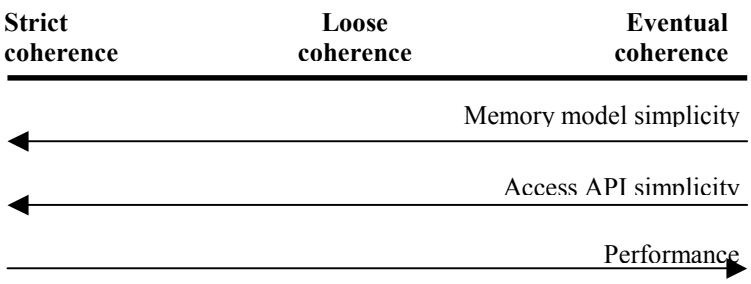

Figure 3: Selecting a consistency model is a tradeoff between performance and memory model simplicity.

Despite the fact that relaxed models are generally considered to be beneficial for performance (see Figure 3) our first approach has been to build a strictly coherent memory system. Two rationales argue for this alternative. Firstly, strictly coherent systems provide a simpler memory representation than a weaker model. Secondly, taking as hypothesis the profile usage model presented in section 3.2, the main type of object is migratory object as defined in Munin[20]. Migratory objects are usually accessed by one process for several consecutive accesses, before another process accesses the object. Munin argues that for this type of object it is efficient to migrate the object on the processing location with full access rights by invalidating other copies. Explicitly, for this sort of data, strict coherence can be handed efficiently by a protocol offering migration capabilities. The protocol we have designed has the advantage of providing for the migration of objects, and we see pro-active migration as a possible means of offering availability of data and coherence despite intermittent connection. At least, we want to test the limit of such a proposition.

[3] Chaotic accesses are non-synchronizing competing accesses occurring when a shared data is accessed in a pattern outside the scope of its associated pattern access model. Limiting the impact of chaotic access imply means for providing a "fairly recent" value. That is, if accesses to variable x are unsynchronised, then reading x must not return any value but a "recent" one. David Mosberger indicates that most of DSM systems have not sorted out this issue.

4.4 Trust and security

Device heterogeneity

In recent years, a wide range of new "appliances" has appeared, and they need to be integrated into larger computing systems. These devices vary in their capacity to protect the data they contain and the interactions the owners of these devices may have with various services offered elsewhere. Some devices are trusted platforms[4], others fall into a large range down to simple devices that offer no security mechanisms beyond keeping data in some local unprotected storage. Authentication, authorization, encryption, monitoring, and many other aspects of the traditional security may not always be possible for these devices.

Level of Trust

In our system, profiles are managed by policies used to control the flow of data to and from the device owned by a person. We have expanded the policies to include trust considerations.

Private information is stored on the devices owned by the user, part of the user trusted domain[5]. This private information is distributed amongst the devices according to the trust level of the device, the user's subjective approval, and on demand. The trust level of the device will depend on factors including the device technical characteristics, its location, and other factors that could be business related. The assembly of conditions are

---

[4] **Trusted Platforms**: Capable to secure the overall system, its interactions, interface, and management and maintain protection of its subsystem capabilities and data. Hence, trusted platforms have built-in security capabilities.

From the Trusted Computing Platform Alliance (TCPA) [23]:

Protection: "describes the properties of selected capabilities and selected data locations within a platform that has a Protection Profile and has not been modified by physical means. A protected capability is one whose correct operation is necessary in order for the operation of the subsystem to be trusted. Protection includes also the concept of shielded (data) locations. The trust in the subsystem depends critically on the access of certain data. Sensitive data should be accessible only to protected capabilities."

A " Trusted Platform enables an entity [a user in particular] to determine the state of the software environment in that platform and to SEAL data to a particular software environment in that platform. The entity deduces whether the state of the computing environment in that platform is acceptable and performs some transaction with that platform. If that transaction involves sensitive data that must be stored on the platform, the entity can ensure that that data is held in a confidential format unless the state of the computing environment in that platform is acceptable to the entity."

Most trusted platforms provide platform integrity, identity, and protected storage.

[5] **Trusted User Domain**: A space that ensures privacy, integrity, and authenticity (all which foster trust) for the user. Such a domain supports strong separation to and from the outside world and controls the respective access.

part of the user context that seems to change dynamically. For our first implementation, we use simple user input to indicate the trust level.

The profile data are partitioned such that sensitive information is available only to the devices that meet the necessary criteria. This information could be encrypted. Some devices could be just temporary holders of the encrypted information, having no means to decrypt it or store it locally for long periods of time. Data may need to migrate immediately, perhaps under some extreme conditions (for instance, if the environment is compromised in unexpected ways).

We need to protect sensitive data under specific threats, all varying with the device characteristics. Solutions include:

- **Encrypt and move data on demand**. If a threat is perceived (attack detected, user indicates entry into non-trusted network etc.) sensitive data may be encrypted and/or moved to a less vulnerable device.
- **Delete data on demand**. For some devices, the most appropriate response to a threat is to remove all copies of the data. This is particularly interesting for all data that exists as shared copies.
- **Filter storage requests to meet storage policies**. When it comes to storage, the user may have definite preferences and polices. Some information may be required to be stored in a certain way, for instance, following some classifications, or for a certain given time, in a certain context, etc.

The focus is not only on the user profile stored on the user's devices, but also on the interaction between the profile and services (directly or via agents). We envision profile platforms and services that filter and control the ultimate use of the profile. This may involve trusted third parties.

Traditional security and trust mechanisms

Traditional security for distributed systems involves authentication and authorization mechanisms that are often dependent on specialized servers. Ideally, the devices and the users behind should have the ability to uniquely identify and finally authenticate before any communication. These devices are expected to operate under peer-to-peer architecture models. Various authorization schemes and data integrity mechanisms may be considered, but these will be restricted to the devices that lack the necessary local resources. In the case of mobile devices, the authorization solution has to scale to a large number of services and has to preserve the user’s privacy. Users may also want to limit the access (in time and space) to private information. There are attempts to classify the information and use it in well-understood contexts[24].

Initially, the architecture opts for traditional security measures in terms of authentication, authorization, private communication links, etc., although we have a distinct preference for lightweight mechanisms, where possible. There are a number of technologies that we consider adapting, like SSL/TLS-based[25][26], as a means of securing connections, TCPA as a means of getting a trusted platform, and Public Key Infrastructure (PKI) or

Pretty Good Privacy (PGP)[27], for authentication and authorization. More details on this line, along with a new proposal, are presented in a paper by Jeff Morgan et al.[28]. Unfortunately from a security point of view, relatively few devices will be TCPA enabled, and this results in a significant security risk for each device, even if the connections themselves can be secured. We also are looking at known specific user and device authentication mechanisms that allow us to validate user identity before permitting access to the device profile system; biometrics, smart cards etc. all apply here. Identity based message encryption also appears to be an interesting approach that could be applied.

Mechanisms to enhance privacy and user control

User-controlled devices contain profiles personalized for a large range of functionality and service access. It seems desirable to allow the owner of a device to dynamically and temporarily disable a class of functionality. This intervention should be simple.

Several approaches are possible. At the simplest, the user may be able to indicate to the device a change in the level of trust; e.g. by pressing a single “don’t trust” key. A more sophisticated approach is possible where a dynamic trust attribute can be calculated as a function of device characteristics and the environment. In either case, the data can be combined with a policy that results in protective measures being taken when the level of trust drops below a specified threshold.

4.5 System design

In this section, we present the coherence protocol that takes in consideration trust and security aspects. The distributed profile forms an object space, where a profile element is an object. The object space is distributed among hosts, i.e. user devices. Several copies of the same object can exist but only one copy is writable: the master copy. Object copies are scattered on different memory hosts. Before writing to an object, a host has to gain control of the master copy. On each host a middleware component handles requests coming from other hosts via the network to support the coherence protocol, manages the local cache and handles request coming from local applications.

The coherence protocol we have initially chosen is a write-invalidate protocol adapted from the COMA-F [29] coherence protocol. The protocol has been changed to reduce communication overhead by storing the sharing list on the master host instead of the home directory, as proposed in SC-COMA[30]. Another modification is conditional object-copy caching: before being authorized to cache a readable or a write-able object-copy a decision algorithm that takes into account the requester host capabilities and data sensitivity is performed. Figure 4 represents the states and possible state transitions of an object-copy located in a cache of one host. For instance, transition (1) corresponds to a read request on an object. An object-copy is cached but has been invalidated as it is in the ***invalid*** state, so the host space manager requests a valid object copy, and as it obtains one, the new state associated with the object-copy in the host cache becomes ***shared***. The current master of an object runs a policy engine to check if the requester host is allowed

to cache the object, transition (1) occurs if the requester is authorized to cache the object; if caching is not authorised, the current master will return a readable value but with no cache rights, transition (9).

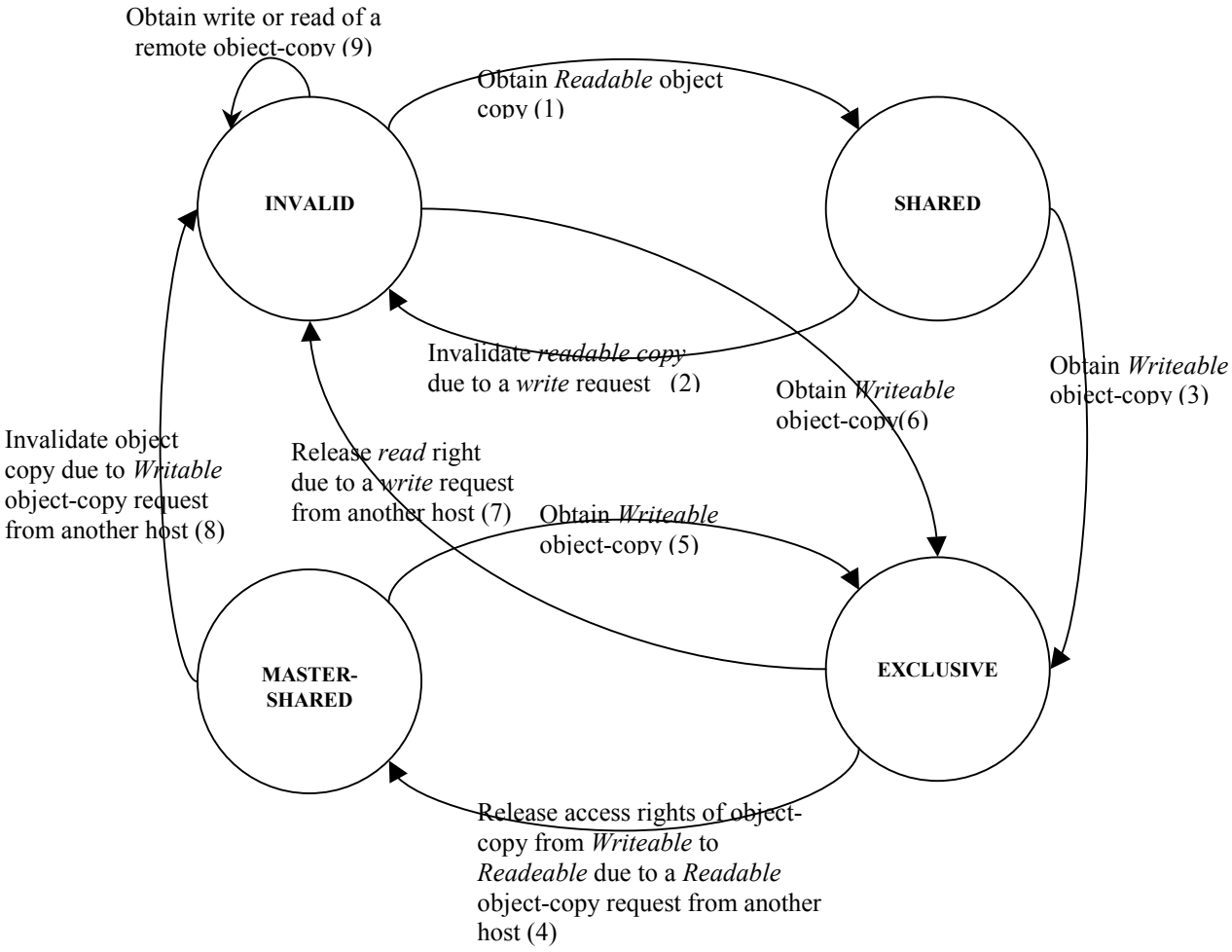

Figure 4: State diagram of the coherence protocol

There are several reasons to limit the access of some hosts to some objects. First, hosts are heterogeneous devices, so some of them have limited power or storage resources. In such a case, it may be inappropriate or even infeasible to cache an object-copy and so object access would have to be achieved remotely, see Figure 5. Secondly, devices are not equally trusted, for example, a mobile device is more prone to be stolen than a PC at workplace, or a PDA may not support the same security capabilities as a laptop equipped

with a smart card reader peripheral. Hence, as a function of the level of data confidentiality, objects may or may not be stored on certain devices. Finally, impeding the migration of an exclusive object-copy on a sporadically connected device is a way to provide higher data availability. In our current implementation, when the master of an exclusive object copy is disconnected from the system, there is no way of getting a guaranteed up-to-date object-copy.

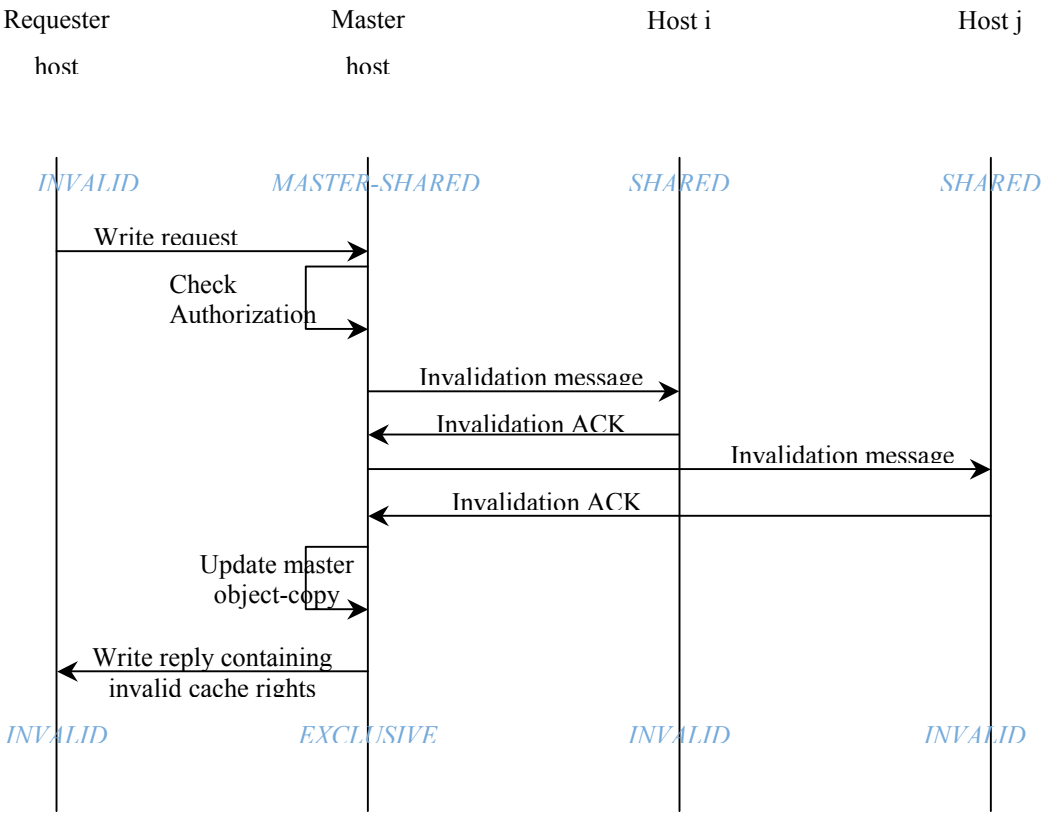

Figure 5: message sequence chart of write request resulting on a write on the current master. The requester sends a write request to the master host. The master host, checks the requester caching rights, the authorization check results in the forbidding of being exclusive master for that object. The writing on the object occurs on the current master host and the requester stays in *invalid* state.

Different decisions can be taken following the defined policy, the object sensitivity attributes and device capabilities:

- Allow caching of object-copy and migration of the master authority (exclusive or shared master).
- Allow caching and migration of the shared-master authority only. The migration of the exclusive master authority is forbidden.
- Allow only caching of a shared object-copy, i.e. no migration of the master authority,
- No caching allowed for this object on this device. Access to a copy of this object will have to occur remotely each time the given host wishes to read or write the given object.

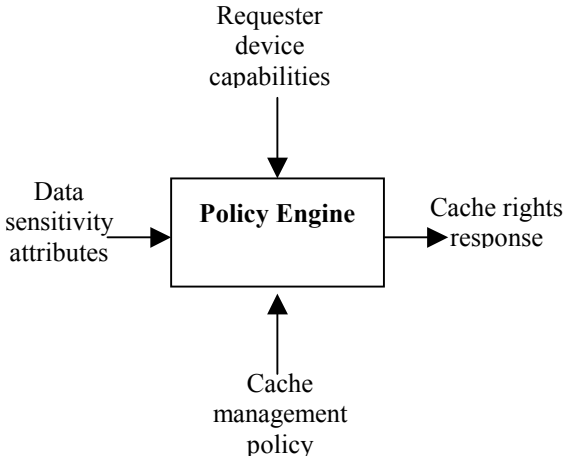

Figure 6: Access policy

5 First version of the research vehicle

We have implemented the system described here by extending Java RMI [31]client server model to a peer-to-peer[6] model. In effect, RMI supports a client/server model of object distribution, one device is a server of objects, and others are clients, when a client wants to get some information it invokes a method on the server object. We propose that several devices share an object and collaborate to handle the coherence protocol and to keep it consistent. When a device needs to get some object value it invokes a method on the local representative for that object which may answer immediately, or collaborate with other representatives for the given object to get an accurate value.

5.1 Flexibility

Our first implementation of a profile storage system reflects our main objectives as explained in previous sections, but we have designed a flexible (object oriented) architecture to get a modular structure. This permits us to replace components of the system (e.g. the coherence protocol) without major architectural changes.

Indeed, a possible direction in the future is to associate specific coherence protocols to specific types of profile data and usage model. However, we currently work on migratory objects, and this implementation only supports the protocol described in paragraph 4.5.

[6] "Put simply, peer-to-peer computing is the sharing of computer resources and services by direct exchange between systems"[32]

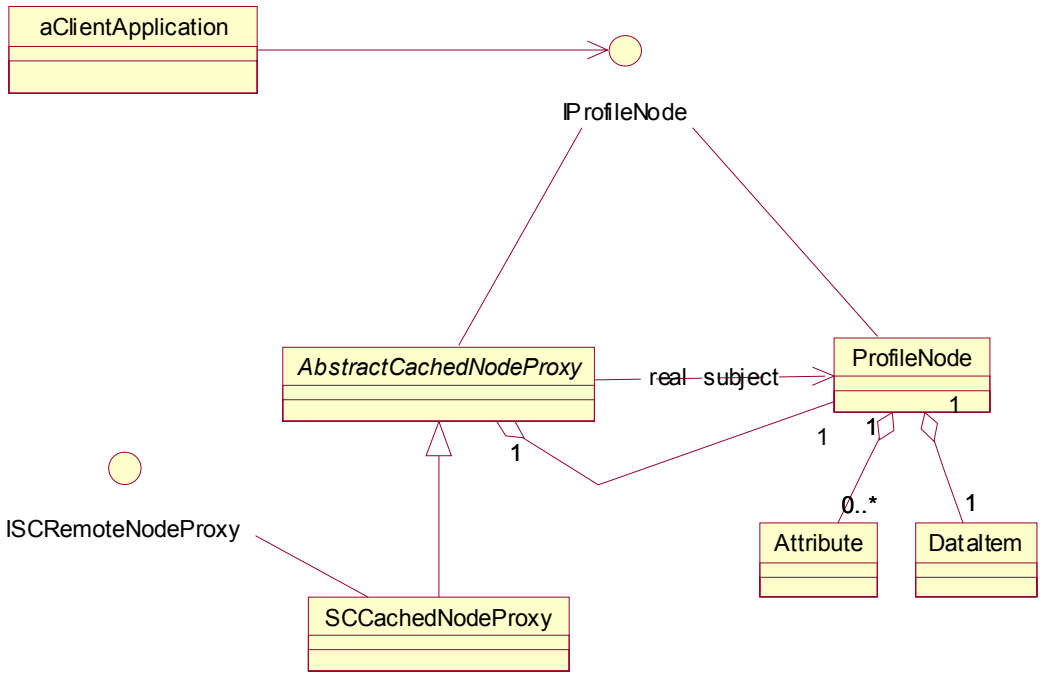

Figure 7: proxy pattern applied to profile node. Each profile node can be handled by a specific protocol depending on the type of proxy that manages it. Participants of the proxy pattern are: INodeProfile: A client application accesses a profile node by invoking an operation proposed by this interface without knowing if an up-to-date copy exists locally or if operations will be executed remotely or if a copy will be recuperate. AbstractCachedNodeProxy: A node proxy implements the node interface so that proxy can substitute ProfileNode. The proxy is also responsible for creating and deleting profile nodes. ProfileNode is the real object that the proxy represents.

Our first implementation stores a hierarchical profile, which is a tree of profile nodes, a node being composed of an item object and a set of attribute objects. Attributes are meta-data that can be interpreted by the profile store, whereas a data item only has meaning at the application level. (A meaningful example of metadata is a sensitivity attribute used by the policy engine to determine caching rights as illustrated Figure 6.) A proxy object, an object inheriting from AbstractCachedNodeProxy (Figure 7), manages a profile node. The proxy object is a surrogate for a profile node object. A proxy controls a client access to the real object. This proxy can serve several roles:

- Cache proxy: it handles coherence requests and keeps a copy of the node;
- Protection proxy: it checks caching rights of a host before sending a copy of the given profile node;
- Remote proxy: it provides a local representative of a remote object that may not be copied locally for different reasons evoked previously.

The first concrete proxy class handles strong coherence between copies of a profile node, SCCachedNodeProxy Figure 7. Adding another coherence mechanism would result in the design of another concrete class inheriting from AbstractCachedNodeProxy.

5.2 Transparency

Access to a profile node by a client application is location transparent. The client application accesses profile node values by invoking method of the IProfileNode interface as if the profile node was local. The proxy provides the identical interface and adds the mechanisms to maintain consistency inside the method implementation of this interface. The proxy collaborates with other proxy of the same object located on other hosts through the ISCRemoteNodeProxy interface.

Each proxy, hence profile node copy, is persistent (i.e. it survives when the local manager of the profile store is stopped) and has a persistent identifier (i.e. when the local manager restarts, the communication between proxies can resume automatically). The latter functionality, provided by RMI activation mechanisms, allows easy reestablishment of communication aong objects after a system shutdown or crash.

This first design and implementation provides no support for disconnection. We intend to make experiments and get knowledge on profile data locality and access patterns. We expect to use this information to get hypothesis on how to take in hand the problem of disconnection.

6 Experiments

The goal of the current research vehicle is to obtain knowledge on distributed profiles, and to determine the suitability of our approach. To this end, we have implemented some simple demo applications designed to allow us to use a distributed profile in what we hope are realistic scenarios.

6.1 Profile manager

The central application is a profile manager that allows a user to observe and manage his profile. The user can browse the profile structure, add and remove profile nodes, and control the sensitivity of profile information by modifying a sensitivity level attribute combined with each profile node.

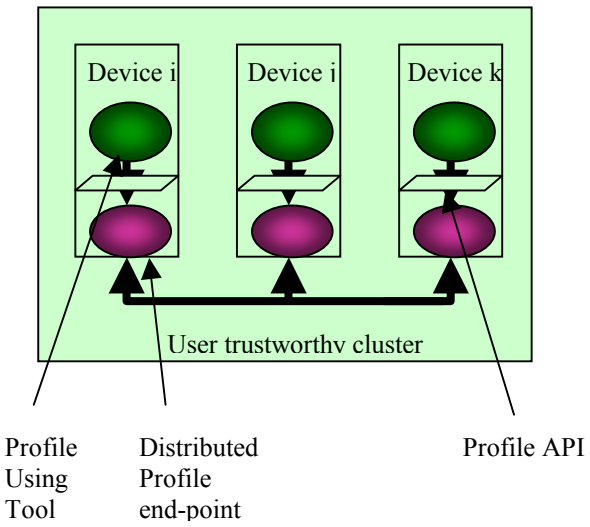

Figure 8: distributed profile architecture. Profile store components are installed in each user device along with the demo tools. These use the profile information stored in the profile end point and accessed through a common profile API.

This application is also a research vehicle to experiment with functionalities that help a user manage his profile.

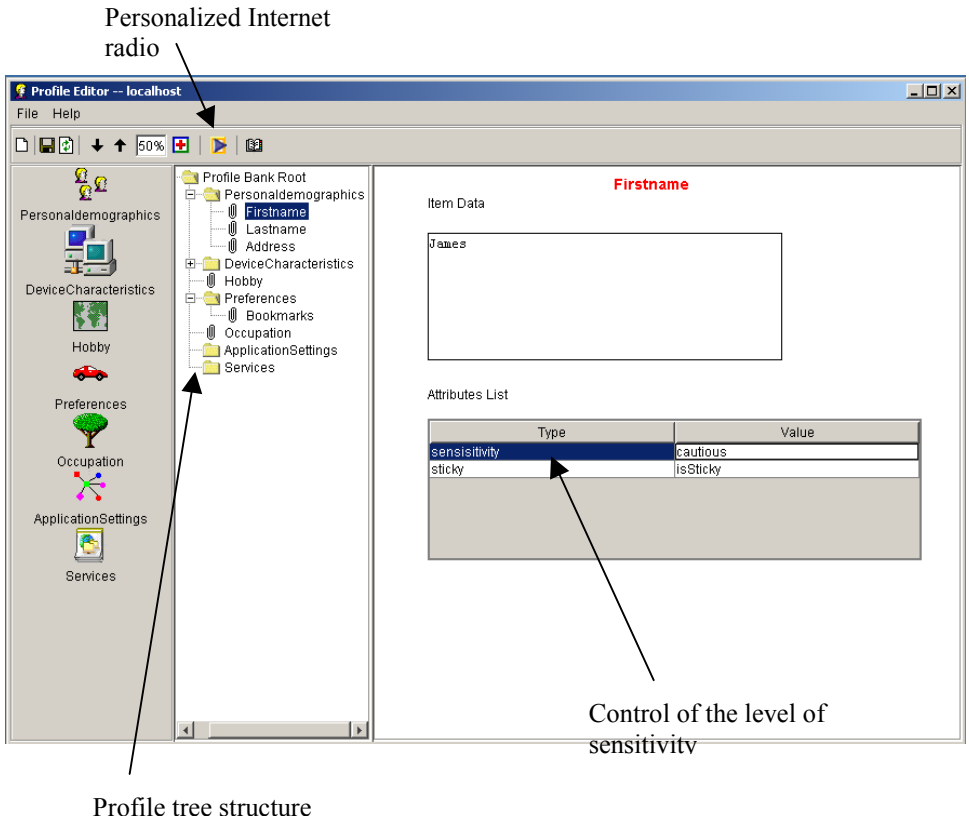

Figure 9: Profile manager. The central application enabling the user to browse and manage his profile.

6.2 Internet Radio

By having a profile store able to store and spread information relative to user habits, user devices can offer personalized interfaces, ideally simplifying the access to information or services. The favourite scenario of our personalized Internet radio demonstration starts with a person at home listening to an Internet radio on a PC/internet radio appliance before going to work. On the way to work, the car radio tunes in to the same radio as was being listened to over breakfast. The program ending during the trip, the user changes station. Once he arrives at his desk, he launches his Internet radio player, which automatically starts on the recently selected radio so that he can listen to the end of the radio program. This scenario is a typical illustration of our use Case 1.

The concept illustrated by this personalized version of an Internet radio is that of a user centred session i.e. user devices that discern user habits, are aware of each other, and collaborate to offer a better user experience. This kind of personalization, the easing of

access to services and information, is a major requirement for helping computer novices enter the digital world.

6.3 Unified bookmark

The idea developed in this demo is to store Internet bookmarks inside the profile store, so that bookmarks collected on one device, are available to all other devices. In addition, these bookmarks can be presented in different manner depending on user context, still with the concern of simplifying access to information. Current solutions[34][35] for having unified bookmarks are server-side, implying that the management of bookmarks is done remotely, not always convenient, lacking in privacy, and with little guarantee of persistence (what happens to your bookmarks if the web site is closed for financial reasons?). This tool illustrates use Case 2.

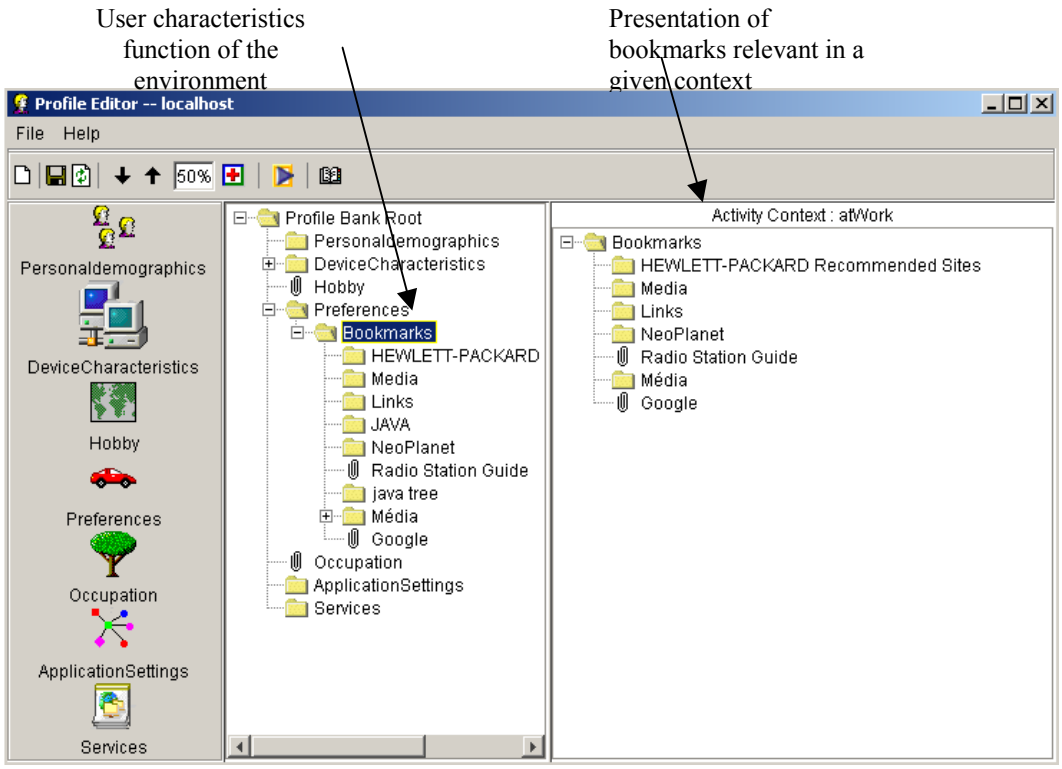

Figure 10: Unified bookmark manager.

On Figure 10, the profile manager plays the role of a bookmark manager offering a contextual representation of user bookmarks. The middle part of the profile manager

displays the whole profile content, hence all the bookmarks, while the right hand side offers a view of bookmarks filtered as a function of the user's context.

The eventual goal of the profile store is to permit any application to get reliable information from the profile store through a standard API.

In addition to these applications, we are designing and implementing a test bench for our distributed profile system that will enable us to "play" out simulated scenarios, and measure system performance. A central test control module will send transaction commands to slave modules on multiple machines, each connected to a distributed profile store end-point in the same way as our example applications. Transaction sequences (involving multiple machines) are defined by scripts, allowing us to replay sequences on different versions of our system. We plan to test the effect of altering cache migration policy, possibly different coherence algorithms, and eventually we will experiment with versions that will tolerate device disconnection.

7 Conclusion

Consumer acceptance of personalized e-commerce and user interfaces depends on the availability of reliable, secure, and especially trusted profile information. We believe that a user's feeling of trust is related to a sense of ownership and control of profile data.

Based on the analysis of some use cases for profile data, we have made the choice of a strictly coherent model, and have taken an approach to trust that restricts the permitted location of data items based on data sensitivity, device capabilities, and a user-specified trust policy. This allows us to deal with the heterogeneity of device security capabilities. The implementation of these models is via the coherence protocol we propose: a write invalidate protocol with dynamic conditional caching rights.

A research vehicle has been implemented in Java using RMI object level services such as persistence and communication management. Further research entails running experiments on this system to collect information on system behaviour when accessed following identified patterns. We foresee that this will permit us to define pro-active migration policies that may enable profile access in an intermittently connected infrastructure.

8 References

[1] K.SCRIBBEN - Privacy@net An internation comparative study of consumer privacy on the Internet- Consumers International. January 2001. ISBN 19023 91 31 68.

[2] MICROSOFT CORPORATION - *Building User-Centric Experiences: An Introduction to Microsoft .NET My Services* - (formerly code-named "Hailstorm")" September 2001. http://www.microsoft.com/net/netmyservices.asp.

[3] LIBERTY ALLIANCE – *Liberty Alliance Project* - http://www.projectliberty.org

[4] J. NEILSON - *Noncommand User Interfaces*- Communications of the ACM 36, 4 (April 1993), 83-99

[5] G. BREBNER - *Matching user needs to product offerings in a friendly way* - In Proceedings of the COST 254 workshop on Intelligent Terminals, Bordeaux, March 23-24, 2000. pp 75-78.

[6] BELARC INC. http://www.belarc.com/

[7] E. RAFFAELE, G. BREBNER - *Consumable Services* - HP Labs Technical Report HPL 2000-161 December 2000.

[8] E. RAFFAELE, G. BREBNER - Process for executing a downloadable service receiving restrictive access rights to at least one profile file - European Patent Application 01410021.8-2201.

[9] A.K. DEY - *Understanding and Using Context* - Personal and Ubiquitous Computing, Vol. 5, 2001.

[10] A.K. DEY, G.D. ABOWD -*Towards a better understanding of context and context-awareness* CHI 2000, Workshop on the What, Who, Where and How of Context Awareness (2000). ftp://ftp.cc.gatech.edu/pub/gvu/tr/1999/99-22.pdf

[11] WORLD WIDE WEB CONSORTIUM – *Platform for Privacy Preferences Specification 1.0* - http://www.w3.org/TR/P3P/

[12] G. COULOURIS, J. DOLLIMORE, T. KINDBERG – *Distributed Systems Concepts and Design* – second edition, Addison-Wesley, ISBN 0-201-62433-8.

[13] STARFISH SOFTWARE, INC – *TrueSync* - Http://www.starfish.com/solutions/solutions.html

[14] PHILIP A.BERNSTEIN, NATHAN GOODMAN – *Concurrency Control in Distributed Database Systems*- Computing Surveys, Vol. 13, No. 2, pp. 185-221, June 1981.

[15] M. SATYANARAYANAN - *Scalable, Secure, and Highly Available Distributed File Access* - IEEE Computer, Vol. 23, No. 5, 1990.

[16] V. SRINIVASAN, J.C. MOGUL - *Spritely NFS: Experiments with cache-consistency protocol*- Proceedings of the Twelfth ACM symposium on Operating Systems principles, 1989.

[17] JOHN K. BENNETT, JOHN B. CARTER, WILLY ZWAENEPOEL, - *Munin: Distributed shared memory based on type-specific memory coherence* - Proc. of the 1990 Conference on Principles and Practice of Parallel Programming.

[18] DAVID MOSBERGER – *Memory Consistency Models*- Operating Systems Review.

[19] M. DUBOIS, C. SCHEURICH, F. BRIGGS – *Synchronization, Coherence, and Event Ordering in Multiprocessors* - IEEE Computer 21, 2 (February 1988), 9-21.

[20] JOHN B. CARTER – *Design of the Munin Distributed Shared Memory System*- Journal of Parallel and Distributed Computing. Special issue on distributed shared memory, 1995.

[21] A. DEMERS, K.PETERSON, M. SPREITZER, D. TERRY, M. THEIMER, B. WELCH –*The Bayou Architecture: Support for Data Sharing among Mobile Users*- Proceedings IEEE Workshop on Mobile Computing Systems & Applications.

[22] B. SCHNEIER - *Secrets & Lies* –Digital Security in a Networked World, John Willey & Sons, Inc., 1963. ISBN: 0471253111.

[23] TRUSTED COMPUTING PLATFORM ALLIANCE -*TCPA Specification V1.1*- http://www.trustedpc.org/home/home.htm .

[24] A. F. LATEGAN, M.S. OLIVIER – *On Granting Limited Access to Private Information* – http://www10.org/cdroms/papers/309.

[25] NETSCAPE - *Secure Sockets Layer* – SSL 3.0 Specification, http://www.netscape.com/eng/ssl3/

[26] INTERNET ENGINEERING TASK FORCE - *Transport Layer Security (TLS)* - http://www.ietf.org/ids.by.wg/tls.html

[27] PGPI PROJECT - *The International PGP Home Page* - http://www.pgpi.org

[28] J.MORGAN, H. MORRIS, V. KRISHNAN, A.A. IVAN - *Secure Web Access in an Environment of Mutual Distrust* - HP Laboratory Palo Alto, HPL 2001-60, March 22nd, 2001

[29] T. JOE - COMA-F: A Non-Hierarchical Cache Only Memory Architecture- PhD thesis, Stanford University, 1995.

[30] A.GEFFLAUT, A. MOGA, J. JEONG, M. DUBOIS - *Design and evaluation of a Software-Controlled COMA* - CENG Technical Report 96-03

[31] SUN MICROSYSTEMS, INC – *Java Remote Method Invocation Specification*- Revision1.7, Java2SDK, Standard Edition, v1.3.0, December 1999.

[32] PEER-TO-PEER WORKING GROUP - *what is peer-to-peer* - http://www.peer-to-peerwg.org/whatis/index.html.

[33] B. PARR – The Next Wave of Internet Users: Forecast and Analysis, 2001-2005- IDC , April 2001, document #24441.

[34] MYBOOKMARKS.COM, INC – *MyBookmarks* - http://www.mybookmarks.com

[35] MAGICALLY, INC - *MagicalDesk* - http://www.MagicalDesk.com