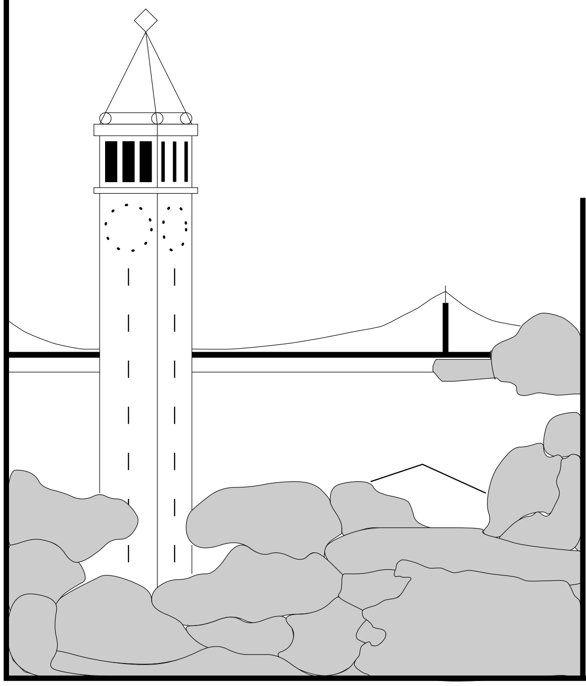
Personal Mobility in the ICEBERG Integrated Communication Network

Bhaskaran Raman   Randy H. Katz   Anthony D. Joseph
{bhaskar,randy,adj}@cs.berkeley.edu
CS Division, EECS Department, U.C.Berkeley



Report No. UCB/CSD-99-1048

May 1999

Computer Science Division (EECS)
University of California
Berkeley, California 94720

Personal Mobility in the ICEBERG Integrated Communication Network

Bhaskaran Raman Randy H. Katz Anthony D. Joseph
{bhaskar,randy,adj}@cs.berkeley.edu
CS Division, EECS Department, U.C.Berkeley

May 1999

Abstract

Communication technology is currently seeing rapid growth, characterized by new access networks, end-devices and an ever growing user-community. However, there is a lack of integration between the different services: a user may have a cell-phone, a pager, a regular phone, a laptop, an e-mail account, and so on. But she has little control over managing communication services in a personalized, unified, and clean fashion across these devices.

The Iceberg Project ([10, 23]) proposes an Internet-Core based approach for integration of telephony and data services across heterogeneous access networks. Personal mobility and personalized communication management as defined by Iceberg are much richer than the corresponding concepts in traditional telephony.

In this report, we present the design of the architectural components that provide personal mobility in the Iceberg integrated communication network. We present our model for a clean, extensible integration of end-devices and networks. We also discuss our mechanism for providing a high degree of personalization of communication. We have a prototype implementation of the integration components in a testbed consisting of four different communication services: GSM cell-phones, Voice-over-IP end-points, Voice-mail, and E-mail.

1 Introduction and Motivation

In today's communication world, there is a wide-range of end-devices (Personal Digital Assistants - PDAs, cell-phones, analog-phones, laptops), access networks (PSTN, GSM cellular network, Pager Networks, the Internet), and services (two-way telephony, multiparty conference, paging, short-message service on cell-phones, voice-mail, e-mail). This list is growing fast: satellite networks, two-way pagers, integrated cell-phone PDAs, instant messaging systems, personal information access services.

In this context, consider the scenario depicted in Figure 1. A user has several communication devices and services: cell-phone, pager, email account, voice-mail service, and PSTN phone at home and office. She wishes to manage communication as follows: she wishes to receive all day-time calls at her office phone and all evening-calls at her home phone. If she is neither in her office nor at home, calls should come through on her cell-phone. However, only certain callers are allowed to call her on the cell-phone; others are redirected to voice-mail.

Further, she wishes to receive headers of emails from people important to her as paging messages. She also should be able to have any email read out to her on her cell-phone by placing a call.

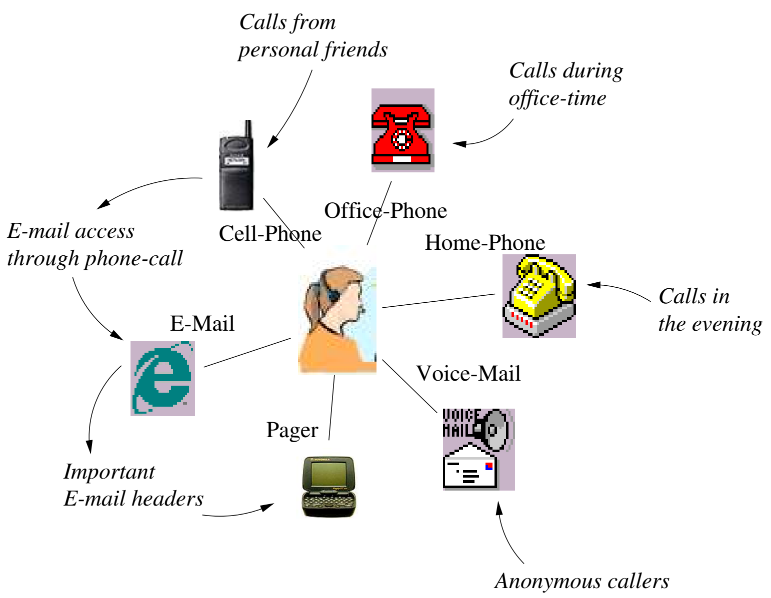


Figure 1: A scenario showing personalized, integration communication

It is clear that the current communication revolution is far from providing such a personalized solution for integrated communication management. While the underlying technology is present, there is a lack of support for complete and meaningful integration of services across end-devices and networks. Even the limited integration present has little or no customizability.

The strong desirability of service integration is proved by the growing number of commercial efforts like email-to-fax, voice-mail-email integration, email-access via cell-phone, etc. (e.g., [15, 19]). While being a step in the right direction, these efforts do not have a model for complete cross-service or any-to-any service integration. They also do not have a good model for scaling to a large user base or to the wide-area.

Efforts in the telephony world for providing integrated communication have been in the form of Personal Communication Services [28]. Several efforts to this end are in the process of standardization. However, since it is built on the Intelligent Network service model [7], these do not provide a flexible service creation environment.

The Internet service model [21] has shown the potential for rapid creation and deployment of innovative services. The Iceberg (Internet-based CorE BEyond the thiRd Generation) project leverages this characteristic to go beyond the service model of traditional telephony. The Iceberg model is shown in Figure 1. The Internet is used as the glue to tie the diverse networks together. This enables two crucial features: (a) a flexible service creation environment in the Internet-Core and (b) a model for cross-network integration of these services.

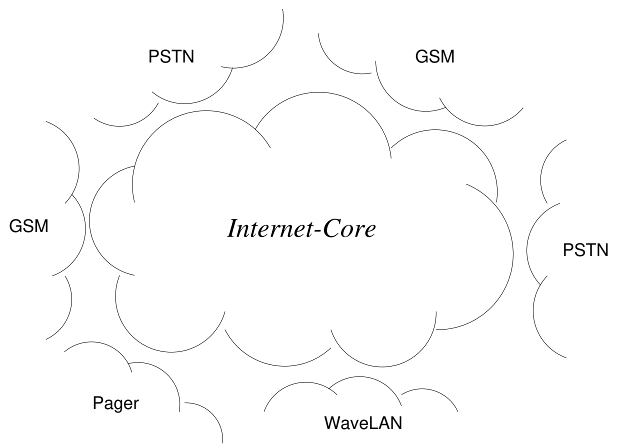


Figure 2: The Iceberg Model: The Internet-core glues the different networks

1.1 Personal Mobility and Customized Services in Iceberg

Personal mobility is one of the important goals of the Iceberg project. The term "personal mobility" is loosely defined and is used to mean different things in different contexts in the telecommunication world. In Iceberg (and in this report), we use the term to mean the ability to have seamless access to services in a network and device independent manner (any service access using any device via any network).

Personal mobility across devices and services is of little use if there's no customizability. The current telecommunication networks provide no customizability or provide it at a very coarse granularity. Personal preferences play an important role especially when we talk about service integration (e.g., which email to convert to pager messages, which incoming calls to redirect to voice-mail and so on). It is personalization that provides the complete solution to meaningful communication management.

A mis-feature of the current communication network is that the caller decides how to reach the callee (except in very restricted scenarios). Personalization goes hand-in-hand with *pushing control to the callee.* Not only does this rid the caller of the difficulty of reaching a person with multiple end-devices, but also, it enables personalized handling of incoming calls by the person being called.

In this report, we describe the design of the architectural components in Iceberg that enable personal mobility. We build the components on top of the *Ninja* [12] service framework. Ninja provides a service execution environment for building incrementally scalable and highly available services. We leverage these properties for our components.

We describe our mechanism for achieving a high degree of personalization and cross-device integration. We have a prototype implementation of the components in a testbed setting. The main piece of our testbed is the interface between the GSM cellular network and the Internet through a GSM base-station. We present our experience in building this testbed, as well as how it is used as a part of the prototype implementation.

1.2 Overview

The rest of this report is organized as follows. In the next section, we present the goals and overall architecture of Iceberg. We also present personal mobility in the context of Iceberg. Section 3 details the design of the main components that are crucial to personal mobility and personalization. In Section 4, we describe our implementation. We present the architecture of the main piece of our testbed – a GSM base-station interfaced to Iceberg. This is followed by a design evaluation and a preliminary performance evaluation in Section 5. Section 6 presents related work. We summarize our work and outline our future plans in Section 7.

2 ICEBERG Goals and Overall Architecture

2.1 Iceberg Goals

The goals of the Iceberg project are as follows.

- *Potentially Any Network Service (PANS):* This goal refers to the ability to access services in a network and device independent manner.
- *Extensibility:* Our architecture should be able to support the easy integration of any new network or service that may emerge in the future.
- *Personal Mobility:* By this, we mean treating the person as the communication end-point rather than a specific device belonging to the user[1]. The mapping

[1] The original idea comes from Personal Communication Services [28]. The Mobile People Architecture [2] also identified this goal.

between a user's multiple end-points is handled crudely and manually in the current communication network. We wish to push this mapping to the communication service layer.

- *Customizability:* The network should provide the mechanisms for enabling end-users to customize their services. Note that this goal is different from one of *usability*. The latter has to deal with the user-interface issue associated with allowing the user to specify a preference profile. In this report, usability is our concern only to the extent that it feeds into the design of the mechanism for achieving the goal of customizability.

- *Ease of Service Creation:* The Iceberg communication network should support a flexible service creation framework. Service creation should not just be restricted to a few network operators as in the telephony world.

- *Incremental Deployment and Scaling:* Iceberg services and the network itself should be incrementally deployable. It should be easy to grow the Iceberg network or increase the capacity of a portion of the network to handle a larger user-community.

- *High availability and Fault tolerance.* Communication plays a very important (and sometimes critical) role in users' lives. The components of the network should be available 24 hours a day and 7 days a week. Iceberg strives for the five nines availability found in traditional telecommunication networks (i.e., 99.999% availability). As such, the architecture should be able to tolerate failures gracefully and hide them from end-users.

- *Operation in the wide-area:* To achieve true global mobility, our architecture should not be restricted to a local area. It should be able to span a large geographic region (across countries or continents).

- *Security and Privacy:* Security, authentication and privacy should go hand-in-hand with personalized communication management. The architecture should support a good mechanism for implementing several security policies.

In this report, we focus on the design of the Iceberg architectural components that are relevant to the goals of personal mobility and customizability. However, throughout this report, the design principles to achieve all of the above goals of Iceberg serve as driving factors. We discuss these principles next.

2.2 Design Principles

- *Network and Device Independence:* This means that our architecture should not be tied to the capabilities of any particular access network or end-device. This is necessary to achieve the goals of PANS and extensibility.

- *Giving control to the callee:* In existing communication networks, the caller decides how to reach the other person. Moving this control to the person being called is crucial for achieving the goal of customizability. This is because customizability ultimately means that a user can manage her communication according to her preference.

- *Well defined service interfaces and composability:* This principle is inherited from the Ninja project (described in detail later). Well defined service interfaces enable easy service composition to add new functionalities. This adds to the ease of service creation. This principle also helps achieve the goal of extensibility − addition of new devices just requires composition with existing interfaces.

- *Multiple administrative domains that inter-operate:* Designing for the model of multiple administrative domains running independent, but co-operating services goes a long way towards achieving the goal of incremental deployment. This principle also helps in wide-area operation. This is best substantiated by the Domain Name Service [26] in the Internet.

- *Hierarchy:* This is a well known principle for distributing a system in the wide-area. We use this principle in the design of the Iceberg components that have to be distributed in order to provide location independence.

2.3 Overview of The Iceberg Architecture

We now present the overall architecture of Iceberg, with special attention to the components that are relevant to personal mobility and personalization.

As suggested by the (expansion of the) name of the project, a fundamental design decision in Iceberg is the use of the Internet as the glue for integrating the various networks. The components that exist in the core IP network are as follows (see Figure 3 for the overall architecture).

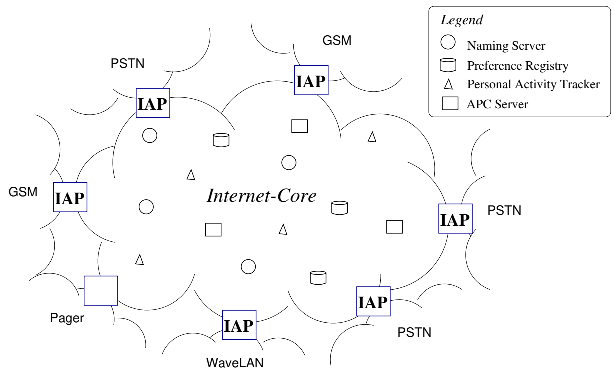


Figure 3: Iceberg Architecture.

- *Iceberg Access Points (IAPs):* The first crucial component in Iceberg is the interface between an access network and the Internet-core network. This component acts as the service transformation agent by providing the impedance matching between device specific functionality and Iceberg services in the Internet-core. It enables the access of services across networks (an important step towards the goal of PANS). These components are called *Iceberg Access Points (IAPs)*.

- *Preference Registry:* To achieve the goal of customizability, we need to have a mechanism for storing and processing user "preference profiles". The *Preference Registry* component of our architecture provides this functionality. The preference registry can be queried to get the user's current preferences (e.g., the current preferred end-point at which to receive calls).

- *Personal Activity Tracker (PAT):* The *PAT* is a component that inter-operates with the user's preference registry. It tracks dynamic information such as the user's current location information or the call state at a particular end-point (e.g., phone is busy, the user is out of GSM cell-phone coverage, etc.). These are used by the preference registry as additional inputs in processing the user preference profile.

- *Naming Service:* A naming service in a system is used for lookups based on names from a particular name space (e.g., DNS [26]). The result of the lookup could be anything: it could be another name, it could be some information associated with the lookup name, etc. In Iceberg, there are different kinds of end-devices we have to deal with: cell-phones, pagers, telephones and so on. Each of these devices have names in different name spaces (e.g., cell-phone numbers, pager numbers, email addresses etc.). In addition to these, we define the notion of a person's *unique identity* which maps across all the devices the person may own. This mapping serves as the first step towards achieving the goal of personal mobility. The unique-ids comprise yet another name space.

  We can have different kinds of lookups based on these names. For instance, we may wish to lookup a person's unique-identity based on her cell-phone number; or, we may wish to find the location of a person's preference registry based on her unique-identity. We capture such lookups based on names in the *Naming Service* component.

- *Data-Path Creation:* To integrate services across heterogeneous end-devices, we need to have a mechanism for performing any-to-any data transformation and transport (e.g., converting from GSM encoded audio to ASCII text). We use the concept of *Paths* and *Automatic Path Creation (APC)* from the Ninja project [12]. The *APC Service* component of our architecture creates paths to handle the data flow and conversion between any two end-points. We elaborate more on this in the following Subsection.

2.3.1 Ninja: Iceberg's Execution Environment

The Ninja project [12] aims to provide a software infrastructure for easy, reliable, and scalable distributed service creation and execution in the Internet. Ninja provides a model for making these services *fault-tolerant*, *highly available*, and *incrementally scalable*[2]. In this section, we briefly present the relevant concepts from the Ninja project and describe how they fit in with the goals of Iceberg.

The Ninja Service Execution Model: The Ninja execution environment consists of three kinds of components: *Bases, Units* and *Active Proxies.* A *Base* is an execution environment on a cluster of commodity workstations located in the network infrastructure [8]. Bases are meant for running services that have to maintain persistent state (state that has to be maintained across transient failures – e.g., users' preference profiles). At the other end of the spectrum are a large number of *Units*: the clients of the services running on Bases (e.g., a laptop, a pager, a Personal Digital Assistant, or a cell-phone is a Unit). In the case where a Unit does not have the intelligence to talk directly with a service on a Base (e.g., a cell-phone which does not know about Bases), an *Active Proxy* is used to reach the Base. In the Ninja model, an Active Proxy has only session state (soft state) which can be restored in the event of a failure. Persistent state exists only at the Bases.

The relation between Iceberg components and the Ninja execution environment is shown in Figure 4. The Iceberg components run as services in the execution environment provided by Ninja. We leverage the scalability and fault-tolerance provided by Ninja's execution environment for our components.

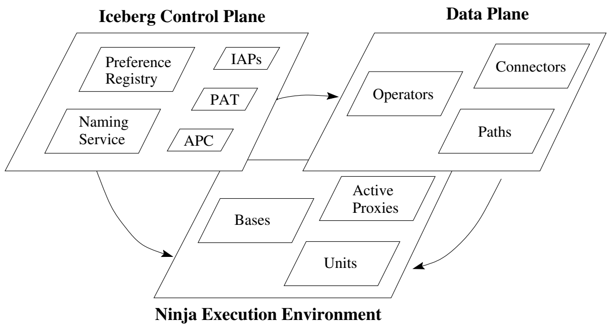


Figure 4: Mapping Between Iceberg Components and the Ninja Execution Environment.

In addition to the execution environment, Ninja strives to provide easy service composability. We use this concept in the data-plane as shown in Figure 4. *Operators* are units of computation that have well-defined interfaces. For example, software that converts PCM encoded speech into ASCII text through a Java RMI interface is

[2]Although Iceberg leverages a lot of crucial features from the Ninja service environment, its control architecture is quite independent of Ninja. Any other service environment that provides similar features could be used instead.

an operator. A *Connector* is an abstraction of the data-flow mechanism between two operators. For instance, we can have *Java RMI* connectors or *UDP/IP* connectors. A *Path* is a set of operators strung together using connectors – an abstraction of a data flow. For instance, a series of codec operators followed by a speech-to-text operator is a path.

We use the concept of *Automatic Path Creation (APC)* extensively in our data-path to achieve the goal of any-to-any data transformation and transport. The well defined operator and connector interfaces make it possible to *compose* paths easily. APC is the process of doing this composition automatically by choosing the right subset of operators and connectors to connect any two end-points.

In Figure 4, the Iceberg control components are instantiated on the Ninja execution environments. The control components also instantiate the data-path components on top of the appropriate execution environments.

3 Design of the Components

In this section, we present the design of the components. Subsection 3.1 briefly presents the design of Iceberg Access Points. We then concentrate on the design of the crucial components for personal mobility and personalization: (a) The Preference Registry and the Personal Activity Tracker in Section 3.2 and (b) The Naming Service in Section 3.3. Subsection 3.4 briefly presents the design of the the APC Service.

3.1 Iceberg Access Points

An Iceberg Access Point, as its name suggests, is the point of attachment of a service or a network to the Iceberg-core. It is a protocol, data, and service transformation agent that provides the network and device independence required for achieving the goal of PANS. For instance, an IAP sitting in front of a text-messaging system like the e-mail service or pager service would enable access to this service via any other end-point: say, a cell-phone. Addition of a new service or a network involves the addition of an IAP to do the appropriate transformations.

An IAP performs Iceberg-specific functions such as preference registry lookups, name service lookups etc. on the Iceberg-core side. On the other side, it provides service- or network-specific functionality. Although it is not present in our current design or implementation, we believe that a clean separation of these two types of IAP functionalities is possible.

In our design, an IAP performs a very lightweight function. This is in contrast to the huge switches and components in the telecommunication network – a single switch could serve hundreds of thousands of users. An IAP's lightweight functionality allows easy and independent deployment in multiple administrative domains in accordance with our principle for achieving the goals of incremental deployment and scaling.

3.2 Preference Registry and the Personal Activity Tracker

The preference registry component is directly associated with the goal of providing customizability. In this section, we describe the issues associated with the design of the preference registry and our initial approach.

The preference registry stores and processes user preference profiles. It acts as a communication management agent on behalf of the user being called – in accordance with our principle of pushing control to the callee.

There are two separate but related issues with user-preference specification: (1) the issue of how the system represents, stores and processes user preferences and (2) the issue of building a reasonable user-interface for the user to specify personal preferences which are then converted into the internal representation form used by the preference registry.

In this report, we focus on the first issue and are concerned with the user-interface issue only to the extent that the internal preference representation is related to the ease of preference specification. However, we wish to stress that the user-interface issue is a very important one, especially when we are talking about complicated services[3].

Examples of Preference Profiles

Before we present out mechanism for modeling user preferences, we give a few specific examples of user preferences that are possible in the Iceberg setting (seamless access to integrated services).

- User Alice has a cell-phone and an office-phone. She wishes to receive all phone calls on her office phone during office hours (on working days) and all calls on her cell-phone at other times. However, all business calls should always go to the office phone or to the office's voice-mail system if she's not available – calls from business clients are never received on the cell-phone.
- User Bob has an email account, a ParcTab device [33] that operates in his office building, and a cell-phone. He wants the headers of e-mails from important people to be read out to him in a push-fashion on his cell-phone. The cell-phone should be used for receiving all incoming phone calls – except when he is in a conference room in the office building (the location information comes from the ParcTab) – in which case, a text message from the caller should appear on his ParcTab.

These examples concentrate on personalized handling of incoming communication. However, the preference registry could also be used for customized access to other services. We use the examples to provide a context for the following discussion.

[3]This issue has been faced in the form of feature interactions (where users don't understand the implications of subscribing to multiple services) even in the limited service model of the Intelligent Network (IN) [31]

Classifying the Inputs to the Preference Registry

A User preference profile is essentially a function of several inputs such as caller-id, time-of-day, or user location. The function's output is the preferred endpoint (e.g., user's cell-phone or email-id).

We classify the inputs to the preference registry into two main categories. The first category is per-call information like caller-id, caller endpoint type, etc. The second input category is more dynamic information, such as the user's location or call state. The user preference profile is processed in the context of these inputs.

The Role of the Personal Activity Tracker

The collection of inputs of the second category – the dynamic inputs – is done by the *Personal Activity Tracker (PAT)*. Figure 5 illustrates the conceptual functionality of the preference registry and the PAT. The PAT could use different techniques for collecting the dynamic information. For instance, in the second example described above, it would collect location information from the ParcTab network. In another example, it could get state information pushed to it by an IAP of the Iceberg network.

The PAT simply collects the information present. Whether or not the information is present or is allowed to be collected are separate issues. The set of personal states that are present and are allowed to be collected (by the user) are user-specific. Perhaps this information can also be stored as part of a user's profile at the preference registry. It is also necessary that any communication between the preference registry and the PAT is authenticated both ways and is over a secure channel. We have not yet completely thought through this portion of the interface between the preference registry and the PAT.

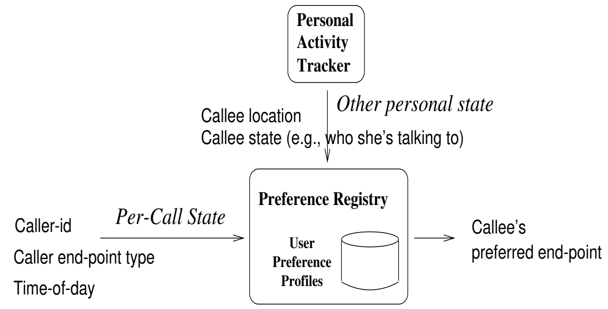


Figure 5: Preference Registry and Personal Activity Tracker

Representing Preference Profiles

The preference profile is a user-specified function of several inputs. A natural way to model it is using a (restricted) scripting language. Given our experience so far, we believe that a rule-based or procedural scripting language can model a wide range of user preferences in a straight-forward manner. Figure 6 shows a rather simplistic example script where the user wishes to redirect communication to her office-phone (9am-5pm), home-phone (5pm-12midnight), or her voice-mail (sleeping time) based upon the time of day. Preference scripts for representing more realistic (and complicated) user preferences such as the examples discussed previously are given in Appendix A.

```
IF (9AM < hour < 5PM)    THEN Preferred-End-Point = Office-Phone;   // At Office
IF (5PM < hour < 11PM)   THEN Preferred-End-Point = Home-Phone;     // At Home
IF (11PM < hour < 9AM)   THEN Preferred-End-Point = Voice-Mail;     // Sleeping
```

Figure 6: A Simplistic Example Preference Script

While a scripting mechanism is very powerful, we should be careful about two issues. The first is that of safety. The language should have minimum possible functionality. Techniques like those used in the *Berkeley Packet Filter (BPF)* [25] are essential to prevent any unbounded execution or infinite loops in the script.

The second issue is a performance concern – having arbitrary code execution during call-setup could mean a lot of additional latency. To address this, we could replace scripting with fast table lookups on the inputs as much as possible. One can imagine such table-lookups based on inputs like the caller-id.

Preference Registry Service on a Ninja Base

The service that implements the preference registry functionality is a crucial piece of the service framework. It could be shared by multiple users in an administrative domain. It needs to be highly available and fault tolerant. Further, it has persistent state: the users' preference profiles. As discussed in Section 2.3.1, a Ninja *Base* is an ideal execution environment for services that have persistent state. Hence a preference registry service would run on a Ninja Base.

Authentication Issues

During a lookup, the preference registry has to be authenticated to the client. Similarly, the client should be authenticated to the preference registry since the caller-id is an important input to the preference registry. Although we have not yet worked out a complete solution, we believe that any authentication has to rely on a public-key infrastructure since we require distributed, wide-area operation. The naming service can possibly be used to distribute public keys as well – we discuss this briefly as part of the next Subsection.

3.3 The Naming Service

Names in Iceberg: Service-Specific-Ids and Unique-Ids

The Naming Service in Iceberg is used for doing any lookup based on a name. We define two kinds of names in Iceberg.

Each of the user's service end-points (or devices) is associated with a **service-specific-id**. For instance, a user could have a telephone number, a pager number and an email-id[4]. To achieve the goal of *personal mobility*, we map all the service-specific-ids of a user to a **unique-id** – which is used to uniquely identify the user in the Iceberg network (this is similar to the Universal Personal Telecommunications - UPT number assigned to a user in PCS).

Name Mapping

One of the important functions of the Naming Service is to map the Iceberg user's[5] multiple devices onto the same logical entity. This is the first step in achieving the goal of personal mobility.

Name mapping is the *lookup* of the *unique-id* based on a given *service-specific-id*. It refers to the mapping from the given service-specific-id to the user's unique-id. This is shown in Figure 7 (we have chosen to have an email-id like name space for unique-ids in this example; however, this choice is not very crucial).

An important point to note is that name mapping is required only when dealing with devices that do not know about Iceberg unique-ids (e.g., a scenario where a user uses her cell-phone to dial another person's telephone number). In such a case, the IAP serving the caller would do the name mapping to get the callee's unique-id. If the calling device is capable of providing the callee's unique-id directly, this mapping is not required. Also note that in some cases, this mapping may result in more than one unique-id (e.g., when the service-specific-id is a shared telephone in an office).

The mapping in the other direction: unique-id to service-specific-id for a particular user, is done at the user's preference registry. This is because the set of devices the user owns and the corresponding service-specific-ids are ideally stored as part of the user's personal profile.

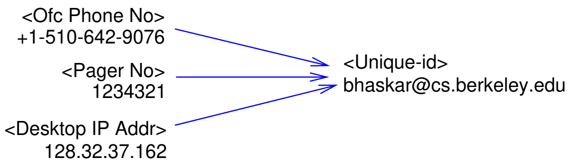


Figure 7: Examples of Name mappings

Using the Naming Service for Preference Registry Location

A second kind of lookup that the naming service currently provides is: the location of the preference registry of a user based on his *unique-id*. This serves as a bootstrap mechanism for learning any information about a user given his unique-id. Further information can be learnt, or a call placed to the user by further querying the preference registry of the user.

[4]We call these "service-specific-ids" and not "network-specific-ids" since a user could have multiple ids for different services in the same network. For instance: an e-mail-id, a Voice-over-IP end-point and an ICQ ID in the Internet.

[5]The term "user" could refer to a human user or any abstract entity in real life – for example, an airline company's customer service number.

Distributing the Naming Service

To satisfy our wide-area requirement, the naming service should be distributed. We need to have a mechanism to locate and retrieve information given a name (service-specific-id or unique-id) in a location independent manner (any lookup from anywhere).

We use the principle of *hierarchy* to distribute the name-mapping functionality. We take an approach similar to DNS: (a) a distributed hierarchical tree of the name space, (b) multiple name servers controlling different portions (sub-trees) of the name tree, and (c) a distributed tree traversal across name servers for name mapping.

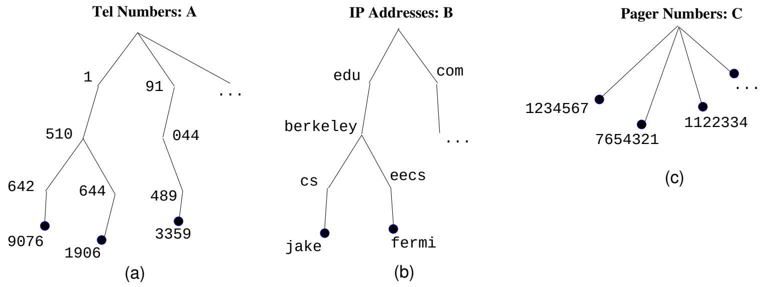


Figure 8: The Hierarchical Name Tree

However, we cannot adopt a DNS-like solution directly since, unlike with DNS, we have multiple name spaces: telephone number space, e-mail address space, pager number space, etc. Furthermore, Iceberg must be capable of handling new name spaces as new services are introduced (e.g., adding a service like ICQ [16] adds a new name space of ICQ IDs).

If we could arrange all the name-spaces into a single tree and have a uniform tree-traversal mechanism, the issue of multiple heterogeneous name spaces is effectively solved. To do this, we first arrange each of the name spaces of the service-specific-ids into a separate tree. How this hierarchical arrangement is done for each name space is orthogonal to the rest of the solution. Examples are shown for three service-specific-ids: Telephone numbers, IP Addresses and Pager numbers, in Figure 8(a), (b) & (c) respectively.

After this first step, we still have the problem of heterogeneous name spaces: hierarchy of strings, hierarchy of numbers split in different ways. We define a schema on each tree. The schema is essentially a definition of: (a) a *tag* associated with each level of the tree and (b) the *type* of the value in that node. The trees in Figure 8 are shown with their tags in Figure 9. As a specific example, the ccode tag in Figure 9 appears in the first level of the telephone-number tree and has values of type numeric.

Now, it is easy to get a uniform view of all trees: a tree's structure is based on its schema. A uniform tree traversal mechanism is possible.

Once we have a uniform view across the trees, it is easy to get a single tree from

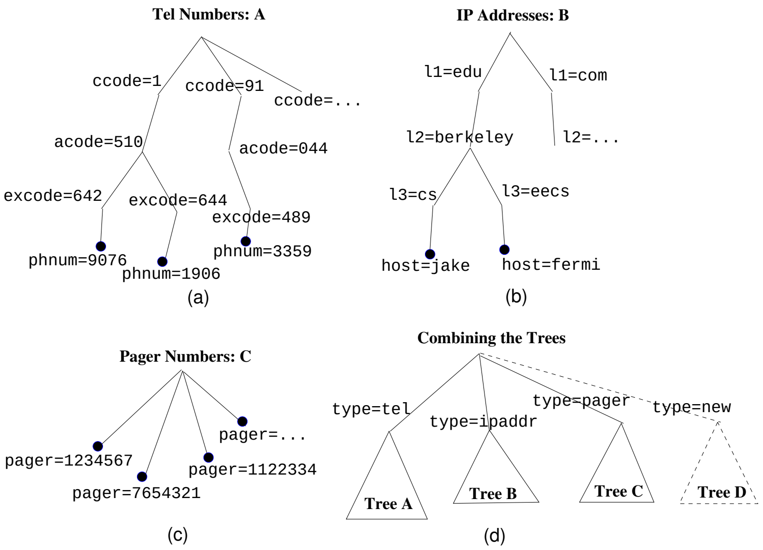


Figure 9: Tagging the Trees

the multiple trees: attach them at a common root (shown in Figure 9(d)). The common root would also store the schema of each of the first level sub-trees. The tree description at the root also allows easy addition of new name spaces by just an addition to the set of entries in the root. The addition of a new name space is shown by dotted lines in Figure 9(d).

The ideas of tags and schemas are originally from LDAP [32].

Given this structure, the rest is similar to DNS. A lookup based on any name can be done by bootstrapping off a name server (configured statically or learnt dynamically using mechanisms such as the one in [3]). After bootstrap, the lookup involves walking the tree (possibly through multiple name servers) to the leaf corresponding to the service-specific-id or unique-id. The leaves (shown as small circles in Figure 9) store the information to be returned in the lookup. The distributed tree enables the use of a distributed and location-independent tree-traversal name mapping mechanism.

Name Server on a Ninja Base

A naming service is a critical component of any system. It needs to be available 24 hours a day, 7 days a week (anyone who has experienced a DNS server crash would readily agree with this). The name server needs to maintain persistent state (across failures). As with the preference registry server, these requirements make it ideal to run a name server on a Ninja *Base*.

Using the Naming Service for Bootstrapping Authentication

The naming service can be used to distribute keys in a public key infrastructure – a user's public-key can be retrieved based on his unique-id. This would be yet another lookup done through the naming service. The naming service itself should be authenticated. For this, a hierarchical authentication mechanism similar to DNS-Security [5] can be used. Note that during a naming service lookup, there is only one way authentication – the client need not be authenticated since we store only public information in the name servers.

3.4 Automatic Path Creation Service

We borrow the design (and implementation) of the APC Service from the Ninja project. Operators and connectors are strongly typed – in accordance with our principle of well-defined interfaces – and can be composed dynamically. This simplifies the addition of any new required data transformations. A new "operator" can be composed with existing ones – only the conversion that is missing could be provided. For instance, suppose that we have the necessary operators and connectors to convert from *PCM encoded voice* to *text* in our system. Now, if we add a new service that requires *text* in a particular *grammar*, the only operator we need to add to the system is one that converts from *text* to that particular *grammar* (possibly using some natural language recognition – if the original voice is from a human speaker).

4 Implementation

To gain first hand experience and to better refine our design, we have been building the components in a testbed setting. The main piece of our testbed is a GSM Base-Station integrated with the Internet-core. Although the GSM "network" in our testbed consists of just two cells controlled by two different transceivers, and is not connected to the GSM wide-area network currently, the testbed is intended to give us insight into aspects of deployment, provisioning, latency and most importantly, usability.

In the next subsection, we describe the GSM-Internet integration architecture and how it fits into the rest of our testbed. We then present the first-cut implementation of the main Iceberg components.

4.1 GSM-Internet Integration Architecture

Our GSM testbed consists of a Base-station Transceiver System (BTS) with two transceivers (TRXs).

The overall architecture of the GSM-Internet integration is shown in Figure 10. The *GSM-IAP* is the component that interfaces the GSM network with the Internet-core. There are two main components that perform the bulk of GSM-specific functionality to interface with the IAP. These are the *UPSim (User Part Simulator)* and the *IP-PAD (IP Packet Assembler & Disassembler)* shown in the figure.

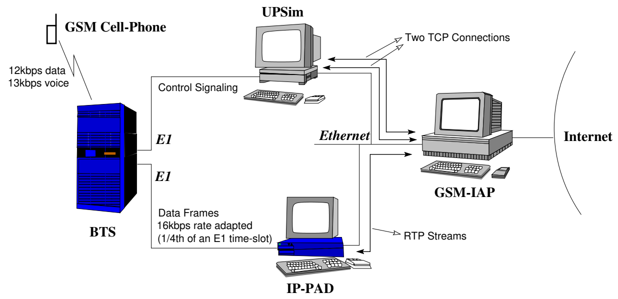


Figure 10: An IAP interfaced to the GSM network

While the UPSim handles control signaling, the IP-PAD, as its name suggests, handles the data-path by inter-converting between GSM frames and IP packets. Together, the UPSim and the IP-PAD provide an Inter-Working Function (IWF) capability. Next, we describe each of these components after briefly describing the base-station.

The Base-Station Transceiver System (BTS)

We have an RBS2202 model BTS with a capacity of six transceiver slots. Of these six slots, two are filled: TRX0 and TRX2. Each TRX can potentially serve a separate cell. Each TRX can handle eight air time-slots. In our configuration, one of the eight time-slots for each TRX is used for control signaling on the air − the other seven can be used for voice or data calls. Hence a maximum of fourteen simultaneous phone-calls are possible in our testbed.

The UPSim

The UPSim is the component that controls the BTS. It handles the *Abis* signaling interface with the BTS thus acting as a BSC (Base-Station Controller) (see [27]). The UPSim also handles MSC (Mobile services Switching Centre) functionality by providing Call-Management and Mobility-Management functions.

The UPSim connects to the BTS via an E1 line. Time slots 1 and 7 on the E1 line are used for signaling to TRX0 and TRX2 respectively (see Figure 11(a)).

The UPSim is a SCO Unix machine with an Ericsson proprietary interface board to the E1 line. It runs proprietary software to handle the signaling with the BTS and the cell-phones through the interface board and the E1 line.

The IP-PAD

The BTS is configured via the UPSim to send and receive all data streams via another E1 line − which connects to the IP-PAD on the other end. Figure 11(b) shows the E1 time-slots used for the data channel. For each TRX, two E1 time slots are used. The data stream rate on air for each air time-slot is 13kbps for voice connections and 12kbps for data connections. This is rate adapted to 16kbps at the BTS and 4 such streams are multiplexed into a single 64kbps E1 time-slot (see Figure 10).

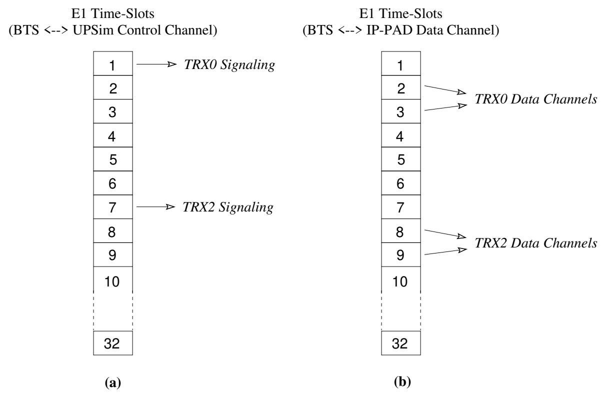


Figure 11: E1 Time-Slots used for signaling and data

The IP-PAD is an off the shelf commodity PC running WindowsNT. It has an E1 interface card[6]. A software module reads/writes frames from/to the E1 line through the card via a driver interface.

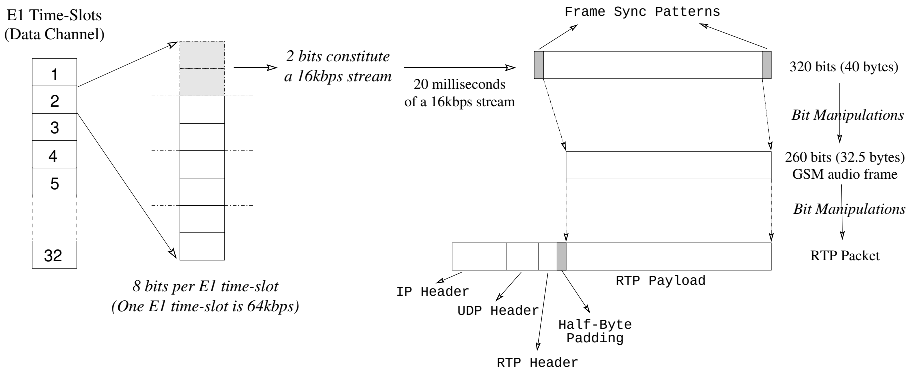


Figure 12: The Conversion of GSM frames to RTP Packets at the IP-PAD

The conceptual operation of the IP-PAD[7] is shown in Figure 12. On reading each of the appropriate E1 time-slots, the four 16 kbps air time-slot streams are demultiplexed as the first step. Then, in each 16kbps stream, the synchronization bits are found and the stream is split into 320 bit frames (one in each 20 ms). Of these bits in a frame, 260 constitute a speech frame in the case of a voice connection and 240 constitute a data frame in the case of a data connection. In the former case, the 260 bits are packed into an RTP [30] packet and sent to the Internet (to the other end point of the voice connection).

In the reverse direction (RTP packets conversion to GSM frames), we need to mul-

[6]Thor-2 Dual T1/E1 Board from Odin TeleSystems Inc. http://www.odints.com/

[7]As of this writing, the IP-PAD handles only voice calls. Support for data calls is not complete.

tiplex four streams belonging to different voice connections onto the same E1 time-slot. In our implementation, we have found this operation to be quite tricky since this involves multiplexing non-isochronous streams from the Internet side onto an isochronous E1 stream. We need to do additional buffering at the IP-PAD in order to collect packets from the four different non-isochronous streams and then send them out as frames on the appropriate E1 slot.

In our testbed, we have used the Internet audio tool `vat` [22] as the other end-point of phone calls to cell-phones. The `vat` tool has a GSM codec option which we use. However, the 260 bits in the GSM frame from the cell-phone do not appear in the same order in the frames used by the codec. We document this mismatch in Appendix B.

The GSM IAP

The GSM IAP is the component that coordinates the UPSim and the IP-PAD. It exports the Iceberg interface to the outside world (the Internet-core) for setting up phone calls to cell-phones and to handle outgoing calls from cell-phones. It implements Iceberg functions like Naming-Service lookup, Preference Registry lookup, Data-path creation through the APC Service and so on.

In our implementation, the GSM IAP is a Java program running on a separate machine and talking to the UPSim via two TCP connections[8]. In our implementation, we have made all the data-streams from the IP-PAD go through the GSM IAP – which then redirects the data-streams via the data-path on the Internet-core side. Of course, the data-streams could be made to bypass the GSM IAP entirely.

The signaling protocol exported by the GSM IAP (to other IAPs) is a simple request-response protocol implemented on top of Java RMI (again for quick implementation). We expect this to change in the future.

The Rest of The Testbed

In addition to the GSM network, the testbed also consists of a WaveLAN network. As of this writing, we are still in the process of adding functionality to our testbed: building IAPs to interface to the PSTN through a H.323 gateway [14] and to interface to the paging network through a 2-way paging gateway. The overall picture of our testbed is shown in Figure 13. The components that we are working on currently are shown with dotted lines. Our immediate plans also include extending the testbed to the wide-area (we are working on deploying Iceberg components at other academic and industrial locations).

4.2 Implementation of The Components

Iceberg Access Points

We currently have IAPs interfaced to four different kinds of services/networks. As

[8] We use TCP connections to the UPSim due to the proprietary software at the UPSim side.

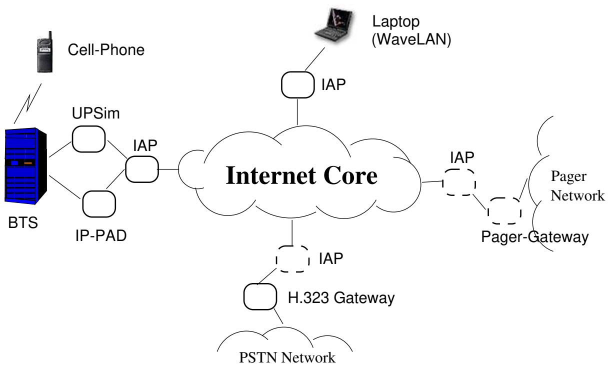


Figure 13: Our Testbed

described in the previous subsection, one of these is the IAP to the GSM network of our testbed. The others are:

- An IAP interfaced to the `vat` Internet audio tool [22]. This IAP is meant to serve a Voice-over-IP end-point: a machine on the Internet. In our setting, this IAP runs on laptops on a WaveLAN network and on desktops on the ethernet − it serves only a single user, the owner of the laptop or desktop. Since the WaveLAN network or the ethernet is just an extension of the Internet-core, there are no special protocol or data transformations to be done for these end-points (in contrast, we required the UPSim and the IP-PAD to do these conversions in the case of cell-phone end-points).
- An IAP interfaced to a voice-mail service. The voice-mail service is also a Voice-over-IP end-point − it is not the one in a traditional telephone system. Thus, this IAP is quite similar to the previous one. No special protocol or data transformation is required in this case as well.
- An IAP interfaced to the user's e-mail inbox. This IAP has to invoke a *text* $\leftrightarrow$ *voice* conversion operator while creating a data path to the other end-point.

It is conceptually easy to add an IAP to the system. Most of the effort in building an IAP goes in interfacing with the access network (i.e., the UPSim and the IP-PAD in the case of our GSM network).

All the IAPs are implemented as Java programs. The main reason for this is that we expect IAPs to run on the appropriate Ninja execution environment (the details of this are being worked on currently). The Ninja execution environments make heavy use of Java's features like strong typing, dynamic class loading, etc. for providing features such as service composition, fault tolerance, and so on.

The Preference Registry

The preference registry is a service running on a Ninja Base. It is a Java program that exports a Java RMI interface for preference lookups. In our prototype, we have concentrated on functionality rather than on performance. We are using a subset of an existing scripting language, perl [18], for the internal representation of user's preference scripts. The Java program invokes the perl interpreter to process the perl script. The Java program and the perl interpreter communicate through the file system.

We are in the process of building and evaluating a user interface for preference specification.

Naming Service

We borrowed the idea of a tagged hierarchical tree from LDAP/X.500 [32, 34]. For quick prototyping, we are using an existing LDAP implementation [17] for the name server. The LDAP server implementation allows us to define the schema of the tree via a configuration file.

LDAP functions, especially LDAPSearch which we use for the lookups, are too heavy-weight for our purpose. This is amply indicated by our high latency measures in Section 5.2. We plan to move to a Java-based implementation of the naming service running on a Ninja Base thus leveraging the scaling and fault tolerance models offered by the execution environment.

APC Service

We are using an implementation of the APC Service, operators and paths from the Ninja project group. This implementation has the operators necessary to support the conversions GSM 06.10 $voice \leftrightarrow text$. However, these operators do not run in real-time. Thus the implementation does not support "stream" connectors which are required for real-time codec conversions.

Personal Activity Tracker

We are in the process of defining the interfaces in the next level of detail for this component. Although not completely interfaced yet, we can currently capture reachability information of users' cell-phones in our GSM testbed by capturing the Location Update, IMSI-Attach, and IMSI-Detach messages from the cell-phones [29].

5 Evaluation

In this section, we present an evaluation of our design (Subsection 5.1) and performance measures (Subsection 5.2). The performance numbers are very preliminary and are not complete. They are meant only to give an indication of possible scaling and latency bottlenecks.

5.1 Design Evaluation

Iceberg is being designed to go beyond the service model of the third generation cellular[9] and PCS (Personal Communication Services) efforts [28, 35]. Its main strength: ease of service creation and deployment comes from the fact that it is designed with the Internet as the core. We now discuss the strengths of our personal mobility model.

5.1.1 Personal Mobility

The concept of personal mobility provided in Iceberg is much richer than the notion of personal mobility provided by PCS. Iceberg provides a framework for cross-network and cross-device mobility of services.

Ease of adding new services

Not only does our model allow for easy introduction of new services (due to the low cost of entry in the Internet), but also it is easy integrate them with existing ones. For instance, in our implementation, it was very easy to add a new voice-mail service and an e-mail service. They required just an additional IAP to interface to the rest of the Iceberg services. For the data-path, they required the addition of $speech \leftrightarrow text$ conversion operators.

Similarly, it would be very easy to integrate other end-points such as PSTN phone, Pagers or even other Internet messaging services, such as ICQ [16].

As an example of an innovative service, consider a "room-control" or smart-space application such as the one described in [9]. Integrating such a service (dynamically) with our system would be as simple as adding an IAP to interface to this service. Users would then be able to access this service via any end-device via any network seamlessly (they could use their cell-phones or pagers to send commands to control the smart devices in a room).

5.1.2 Flexibility in the Personalization Model

In our personalization model, it is trivial for users to have common IN and PCS services. A few examples are:

- Call redirection – our system allows this feature to be based on a variety of user-specified parameters.
- Personal numbering – this involves a simple preference registry lookup in our system.

[9] That's where the phrase "Beyond the Third Generation" in the expansion of "Iceberg" comes from.

- 800-number like redirection – this is just a specific case of call redirection based on factors like caller-id and/or location.

Commercial service integration efforts such as voice-to-email, email-to-fax, etc. are just specific instances of PANS. They are easy to build, deploy, and personalize in Iceberg.

5.2 Preliminary Performance Evaluation

We have gone through this first-cut implementation of the components in the spirit of the *"Plan to throw one away"* system design principle [24]. The performance evaluation in this section is preliminary and is intended to throw light on what the potential latency bottlenecks could be.

A natural result of the extra functionality we provide in terms of personal mobility is the addition to the latency in call setup time. This is what we concentrate on in our performance measurements below. We measure only the IAP-to-IAP call setup latency in the Iceberg-core – not the end-to-end latency, which would also include the latency on the access network side.

5.2.1 The Setup

The setup we use for our tests is shown in Figure 14. We use the same machine (labeled S) as the preference registry server and as the local name server. In our configuration, there are no name server referrals – all requests are served by the local name server.

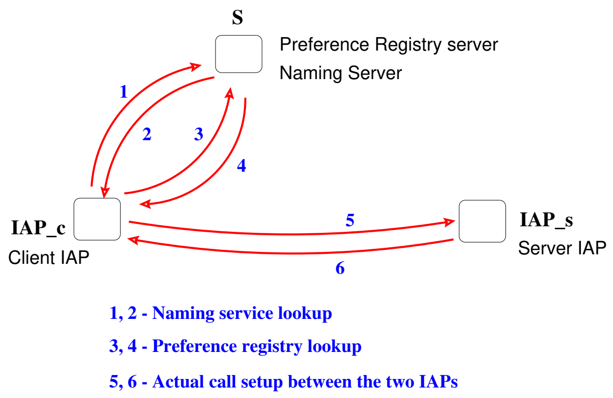


Figure 14: The Setup for Latency Measurements

The IAP client (the caller end), labeled IAP_c, and the IAP server (the called party), labeled IAP_s, are the other two machines in the configuration. A call setup involves

Naming lookup latency	29ms (4)
Pref Reg lookup latency	96ms (11)
Latency of actual call setup	
• RMI lookup for IAP call	22ms (4)
• RMI call between IAPs	46ms (3)

Table 1: Latencies of the Steps in Call Setup

Reading the preference script	2ms (0)
Perl interpreter execution	39ms (11)
Communication with perl interpreter	11ms (0)

Table 2: Latencies of the Steps at the Preference Registry Server

the following steps: (a) naming service lookup by IAP_c to do name mapping and to get the location of the preference registry, (b) preference registry lookup to get the preferred end-point, IAP_s, and (c) the actual call setup between IAP_c and IAP_s.

Both the client and server IAP serve voice-over-IP end-points: the `vat` tool on the respective machines. In this simple case, the APC service is not used at all – there is no data transformation and the data flow is just an RTP stream[10].

The IAP client and server are Pentium 200MHz machines running FreeBSD-2.2.7. The machine S is a 233MHz IBM Thinkpad 560X, running Linux-2.0.36. The Java programs were compiled and run using the *Java Development Kit 1.1.7*. All the machines were on the same 10Mbps ethernet subnet. We have not performed any wide-area tests yet. The results are summarized in Table 1. The figures in parentheses give the standard deviations. In all our measurements, we take the average of ten runs – after ignoring the first measurement which is highly skewed due to dynamic class loading in Java.

5.2.2 Naming lookup Latency

As mentioned earlier, we are using an existing LDAP implementation for the schema based tagged trees. We are using Netscape's directory SDK 3.0 for Java on the client side. Our measurements show latencies of the order of about 30ms for the Naming lookup. We believe that this huge latency is due to the LDAPSearch at the LDAP server – which is a heavy weight operation. This is not really needed and conceptually, our naming lookup should not be any costlier than a DNS lookup which is of the order of a millisecond in a LAN.

[10]Ideally, the APC service would still be contacted to setup an empty *path* – but this is not the case in the current implementation. The APC service did not have support for stream data flows at the time of this implementation.

5.2.3 Preference Registry lookup Latency

We are using naive mechanisms in our preference registry implementation currently. Our measurements show on the order of 100ms latency for the lookup. The bulk of this latency comes from invoking the perl interpreter from the preference registry server – a process execution – and because of the use of files to communicate between the two processes. A good amount of latency also comes from the Java RMI lookup and the RMI call.

Table 2 shows the latencies of the steps at the preference registry server: (a) accessing the stored preference script from a file, (b) invoking the perl interpreter as a subprocess, and (c) communication with the perl interpreter through the file system.

5.2.4 Latency of the Actual Call setup

The actual call setup is just a simple request-response protocol. The latency of this step involves a Java RMI lookup at IAP_c and the RMI call to IAP_s. Both of these steps show on the order of tens of milliseconds latency (Table 1).

6 Related Work

A wide assortment of commercial products that attempt to provide integration across services and communication devices are becoming available. A few examples are: e-mail-to-fax services [11, 15], voice-e-mail-fax integration services [13, 19], and enhanced telephony services [20]. These commercial services show the strong desirability of having personalized integrated communication. The TOPS architecture [1] uses a "directory service" component for personalization. However, the commercial efforts, as well as TOPS, are limited in terms of any-to-any service integration and dynamic service composition. They also lack models for scaling to the wide-area or to a large user base.

The Mobile People Architecture (MPA) [2] has some goals similar to Iceberg. Both projects try to provide any-to-any integration and personalized call handling. However, there are many enhanced features of our design that are missing in MPA: seamless service integration and a model for scalable and incremental wide-area service deployment (a feature that we leverage from Ninja).

The term personal mobility is used to mean different things in the telecommunication world. In the context of GSM, the SIM (Subscriber Identity Module) provides a limited form of "personal mobility" by making the user's identity (the SIM card) independent of the end-device (cell-phone) in use. More recent *Personal Communication Services (PCS)* efforts take this a step further by adding another level of mobility. The user is identified by a unique number – called Universal Personal Telecommunications (UPT) number (much like our unique-id) across multiple devices on fixed as well as mobile networks [28].

While there are similarities in the goals and motivation behind the PCS efforts, there are crucial differences. Our model, since it centered around the Internet, presents a flexible service creation and integration model – service creation is not restricted to network operators (this is the same as the "Boost to Innovation" argument in [21]). It is easy to incrementally deploy and scale our system in the wide-area.

7 Conclusions & Future Work

In this report, we have presented the architecture and design of the components for personal mobility as defined by Iceberg. Our model allows seamless access to communication services and thus provides true tether-less mobility. The ease of introducing and integrating new and innovative services is unique to our design and is missing in the personal mobility efforts in the telecommunication world.

We add a flexible customization model on top of personal mobility to enable personalized communication management. It is easy to enable PCS services and beyond in this model.

We have an implementation of our model in a testbed setting. The testbed consists of cell-phones in a GSM network. The main advantage of this testbed implementation would be a real-life usability and performance study of our model.

We have plans to extend the testbed to a wide area setting to give us insight into the performance and latency bottlenecks in the system. Another crucial aspect that requires study is the usability of the system – especially user-centric aspects like preference specification. As of this writing, we have not done any usability tests or detailed performance measurements (beyond those presented in Section 5.2). We expect to do these studies in the immediate future and refine our design based on the experience.

Acknowledgments

Several people have been a part of different aspects of this project. We are thankful to all of them. The work on the GSM testbed was done along with Sreedhar Mukkamalla and Jimmy Shih – we're indebted to them for the exciting experience. The development of the testbed and its maintenance would not have been possible without the technical help from our Ericsson contacts: Reiner Ludwig, Krister Andersson, Stefan Mohlin, and Kent Larsson. We thank Keith Sklower and Brian Shiratsuki for their continuing technical support with the testbed. The lively discussions with Helen J. Wang, Ben Zhao, and Emre Kiciman have helped shape a lot of the design – we are grateful to them for that.

References

[1] N. Anerousis, R. Gopalakrishnan, C. R. Kalmanek, A. E. Kaplan, W. T. Marshall, P. P. Mishra, P. Z. Onufryk, K. K. Ramakrishnan, and C. J. Sreenan. The TOPS Architecture for Signaling, Directory Services and Transport for Packet Telephony. In *The 8th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, 1998.

[2] Guido Appenzeller, Kevin Lai, Petros Maniatis, Mema Roussopoulos, Edward Swierk, Xinhua Zhao, and Mary Baker. The Mobile People Architecture. Technical report, Stanford Computer Science Lab CSL-TR-99-777, Jan 1999.

[3] Steven Czerwinski, Ben Y. Zhao, Todd Hodes, Anthony Joseph, and Randy H. Katz. An Architecture for a Secure Service Discovery Service. In *(To Appear) Fifth Annual International Conference on Mobile Computing and Networks*, August 1999.

[4] Jutta Degener and Carsten Bormann. GSM Codec Library, University TU-Berlin, FB-Informatik. jutta@cs.tu-berlin.de, cabo@cs.tu-berlin.de.

[5] D. EastLake and C. Kaufman. *Domain Names System Security Extensions IETF. Request for Comments: 2065*, Jan 1997.

[6] ETSI. *GSM Recommendation 06.10.*

[7] Nadege Faggion and Cegetel Thierry Hua. Personal Communications Services Through the Evolution of Fixed and Mobile Communications and the Intelligent Network Concept. *IEEE Network*, Jul/Aug 1998.

[8] Steven D. Gribble, Matt Welsh, Eric A. Brewer, and David Culler. The MultiSpace: an Evolutionary Platform for Infrastructural Services. In *Usenix Annual Technical Conference*, June 1999. Monterey, CA.

[9] Todd Hodes, Randy H. Katz, E. Servan-Schreiber, and Lawrence A. Rowe. Composable Ad hoc Mobile Services for Universal Interaction. In *Proceedings of The Third ACM/IEEE International Conference on Mobile Computing*, Sep 1997.

[10] http://iceberg.cs.berkeley.edu/. *The Iceberg Project.*

[11] http://info.ox.ac.uk/fax/. *Email to Fax at Oxford University.*

[12] http://ninja.cs.berkeley.edu/. *The Ninja Project.*

[13] http://planetarymotion.com/. *Planetary Motion's CoolMail Service.*

[14] http://www.databeam.com/h323/h323primer.html. *A Primer on the H.323 Series Standard.*

[15] http://www.fax away.com/. *Faxaway: The Premier Email to Fax Service.*

[16] http://www.icq.com/. *The ICQ Internet Chat Service.*

[17] http://www.openldap.org/. *OpenLDAP*.

[18] http://www.perl.com/. *The Perl Scripting Language*.

[19] http://www.thinklink.com/. *ThinkLink*.

[20] http://www.wildfire.com/. *Wildfire*.

[21] David S. Isenberg. The Dawn of the Stupid Network. *ACM Networker 2.1*, pages 24–31, February/March 1998.

[22] Van Jacobson and Steven McCanne. VAT Mbone Audio Conferencing Software. ftp://ftp.ee.lbl.gov/conferencing/vat.

[23] Anthony Joseph, B. R. Badrinath, and Randy H. Katz. A Case for Services over Cascaded Networks. In *First ACM/IEEE International Conference on Wireless and Mobile Multimedia (WoWMoM'98), Dallas Texas*, October 1998.

[24] Butler W. Lampson. Hints for Computer System Design. In *Proceedings of the 9th ACM Symposium on Operating Systems Principles*, October 1983.

[25] Steven McCanne and Van Jacobson. The BSD Packet Filter: A New Architecture for User-level Packet Capture. In *Proceedings of the 1993 Winter USENIX Technical Conference*, Jan 1993.

[26] P. Mockapetris. *Domain Names - Implementation and Specification. IETF. Request for Comments: 1035*, Nov 1987.

[27] Michel Mouly and Marie-Bernadette Pautet. *The GSM System for Mobile Communications.* Cell & Sys, 1992.

[28] Raj Pandya. Emerging Mobile and Personal Communiction Systems. *IEEE Communications Magazine*, June 1995.

[29] Moe Rahnema. Overview of the GSM system and protocol architecture. *IEEE Communications Magazine*, April 1993.

[30] Henning Schulzrinne, Stephen L. Casner, Ron Frederick, and Van Jacobson. *RTP: A Transport Protocol for Real-Time Applications. Request for Comments: 1889 – Audio-Video Transport Working Group*, Jan 1996.

[31] S. Srinivasan. Impact of Enhanced Feature Interactions. *IEEE Communications Magazine*, Jan 1995.

[32] M. Wahl, T. Howes, and S. Kille. *Lightweight Directory Access Protocol (v3). IETF, Request For Comments 2251*, Dec 1997.

[33] Roy Want, Bill N. Schilit, Norman I. Adams, Rich Gold, Karin Petersen, David Goldberg, John R. Ellis, and Mark Weiser. The PARCTAB Ubiquitous Computing Experiment. Technical report, Technical Report CSL-95-1, Xerox Palo Alto Research Center, March 1995.

[34] ITU-T Recommendation X.500. *The Directory: Overview of Concepts, Models and Service*, 1993.

[35] Mohammed Zaid. Personal Mobility in PCS. *IEEE Personal Communications*, Fourth Quarter 1994.

A Example Preference Scripts

Preference scripts for the example scenarios in Section 3.2 are shown below. The first figure shows Alice's preference profile and the second one shows Bob's preference profile.

```
IF (Caller-Id IN Set-of-business-clients)
THEN Preferred-End-Point = Office-Phone;

IF (Day-Of-Week IN {Mon, Tue, Wed, Thu, Fri})
   AND (9AM < Hour-Of-Day < 5PM)
THEN Preferred-End-Point = Office-Phone;

DEFAULT Preferred-End-Point = Cell-Phone;
```

Figure 15: Alice's Preference Profile

```
CASE (Incoming-Type == E-Mail)

  IF (From-Header IN Set-Of-Important-People)
  THEN Preferred-End-Point = Cell-Phone;

  DEFAULT E-Mail;

CASE (Incoming-Type == Voice)

  IF (Location IN Conference-Room)
  THEN Preferred-End-Point = ParcTab;

  DEFAULT Cell-Phone;
```

Figure 16: Bob's Preference Profile

B GSM Codec used by VAT

The GSM Codec is implemented in `vat` using the library from TU-Berlin, FB-Informatik [4]. The bits in the speech frames from the cell-phone are not in the same order expected by this codec. Specifically, each of the codec fields: *LPC Filter, LTP Filter & Excitation Signal* (see [27] Section 3.3.2 and [6]) are reversed in bit order. That is, when the 260 bits of a speech frame are ordered $b_1, b_2 ... b_{260}$, each of the fields have to be reversed before feeding into the GSM codec used by the library (for example, the first six bits $b_1 - b_6$ represent the first LPC filter parameter and have to be reversed as $b_6 ... b_1$).