Universal Inbox: Providing Extensible Personal Mobility and Service Mobility in an Integrated Communication Network

Bhaskaran Raman   Randy H. Katz   Anthony D. Joseph
CS Division, EECS Department, U.C.Berkeley, CA 94720, USA
{bhaskar,randy,adj}@cs.berkeley.edu
Phone: +1-510-642-8284, Fax: +1-510-642-5775

Abstract

Communication technology has been seeing rapid growth, characterized by new access networks (e.g., cellular, pager, wireless-IP) and end-devices (e.g., PDAs, two-way pagers, multi-model access devices). There have been several efforts at integrating services across such heterogeneity. However, little work has been done on identifying an underlying architecture for such an integration. We identify the requirements for this in the context of an integrated network with heterogeneous end-points. The **Universal Inbox** *provides (a) generic data type transformation, (b) customizable redirection of incoming communication based on user preference profiles, and (c) device name mapping and translation. We present an architecture mapping these functionalities to reusable infrastructure components realized as Internet services. The unique feature of the architecture is its extensibility – it allows not only the integration of existing end-points, but also extension in terms of the end-devices and novel services it can handle.*

We have implemented the Universal Inbox components in a test-bed setting, supporting a variety of devices and services: GSM cellular-phones, voice-over-IP end-points, voice-mail, e-mail, instant messaging service, etc. With our architecture, building personal mobility and service mobility features, and extending them to new end-points has been easy in concept and in implementation. The performance analyses with the initial implementation show that even the heavy-weight components can be scaled to accommodate a large user-base.

1. Introduction and Motivation

The concept of Personal Communication Services (PCS) comes from the telecommunications domain [9, 26, 31]. While an often loosely used term, PCS, in its most general sense refers to the provision of personalized voice and data services independent of the network [4]. It identifies three kinds of mobility: (a) personal mobility – the ability to redirect communication across heterogeneous user devices, (b) service mobility – this provides access to services independent of the user's end-point; i.e., the user sees the same set of services from all end-points, and (c) terminal mobility – allowing users to move from one physical location to another while having the same set of services available. Terminal mobility is provided by today's cellular networks. In this paper, we concern ourselves with the other two: *personal mobility* and *service mobility*. In the rest of the paper, the term "mobility" refers to these two kinds of mobility.

With new communication devices and services emerging at a rapid pace, today's user has a range of communication end-points. Consider the scenario depicted in Figure 1. A user has several devices (cell-phone, pager, PSTN phone, desktop at office, etc.) and services (e-mail, voice-mail, instant messaging, information access services). She may wish to manage incoming communication in a flexible manner. The figure shows redirection to different device end-points based on who is calling, the time-of-the-day, importance level, etc. Furthermore, the user may wish to access different services: news headline service, personal information like calendar service, her e-mail folders, etc. She should be able to do this independent of the access device she is using.

The first aspect (flexible redirection) in the above scenario constitutes personal mobility. The second (service access) represents service mobility. There has been a lot of interest and previous work on the functional aspects of such features (e.g., [22, 28, 1]). There are also several commercial services that provide some of the integration functionality (e.g., [14, 10, 12, 15, 17]). However, there has been little research into the infrastructure support required to support these mobility features in an *extensible* and *scalable* fashion. There are three main goals in this work:

1. *Extensibility:* With the pace at which technology is moving in this domain, it makes little sense to provide

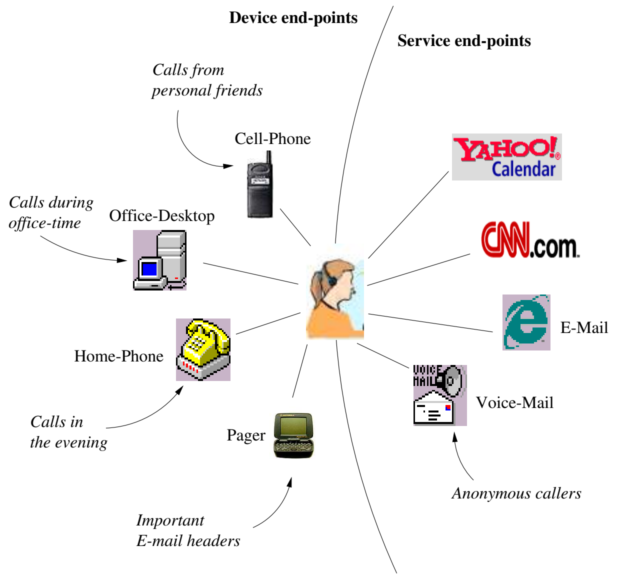

Figure 1. A scenario showing personalized, integrated communication

a mechanism that only accommodates existing devices or services. Thus, our primary goal in the design of the Universal Inbox is *extensibility*. It should be easy to add emerging communication services, and integrate them with *all* of the existing ones. The right component functionalities should be provided so that this process of integration involves minimal development, and minimal deployment.

2. *Scalability:* This is an important concern – the system must scale to accommodate a large user community, and must be capable of operation on a global scale. This is our second important goal.

3. *Personalization:* Integration is of little use without personalization. Specifically, we mean that user should be able to specify preferences for ubiquitous redirection of incoming communication (such as those in Figure 1). This is our third goal.

The **Universal Inbox** is our realization of an integration architecture with all these features. It signifies the mechanism for achieving flexible personal mobility across heterogeneous communication services. The user sees a **unified**, conceptual **inbox** of incoming communication received via different channels. We present a generic architecture for providing *any-to-any* integration of existing as well as future devices. The architecture also supports service mobility across the user's set of devices.

The challenge in the design of the Universal Inbox is in coming up with the right separation of functional components; one that enables a scalable architecture that can easily be extended to newly emerging devices. The components should be independent of one another in design and deployment, while still being inter-operable.

For the realization of the feature set exemplified by Figure 1, we identify three key functional capabilities: (a) any-to-any data transformation – for accommodating diverse devices with different data formats, (b) storage and processing of user preferences – for ubiquitous redirection of incoming communication, and (c) device name translation and mapping – to handle the heterogeneous name spaces like cell-phone numbers, pager numbers, IP-addresses, etc. Importantly, we present an architecture mapping these capabilities to independent *infrastructure* services.

To gain insight into the requirements for building personal mobility services, we have implemented the components of our design in a test-bed setting that includes GSM cell-phones, desktop IP end-points, regular e-mail, instant messaging service, etc. The (re)use of the three functional building blocks made the addition of each subsequent end-point easier. In this paper, we present the design of our system and the experience gained from the implementation. To address provisioning and scaling concerns of the infrastructure services, we have done preliminary performance tests with our implementation. These show that even the heavy-weight data transformation component can be scaled to a large-user base. This supports our architectural decision to place the components in the infrastructure as shared services.

The rest of the paper is organized as follows. The next Section presents the principles underlying our design followed by the actual design of the architectural components. Section 3 illustrates the extensibility aspects of our design by means of a discussion of example services that we have built. Section 4 presents the results of the performance tests for scaling. Section 5 discusses related work and Section 6 summarizes our conclusions.

2. Design of the Architecture

The following principles are key to the design of the Universal Inbox.

1. *Separation of Functionality:* The first principle that we follow is to clearly separate functional capabilities so that they can be implemented and deployed as independent, reusable network components. The reuse of functionality improves extensibility. Further, shared network services can be scaled well using cluster computing platforms [6].

2. *Provide network and device independence:* By providing these, it becomes easy to implement uniform functionality that spans heterogeneous user devices. For

instance, suppose that we have the functionality of e-mail access through cell-phone. With network and device independence taken care of by the components, the same functionality would be readily available on the user's PSTN phone as well.

3. *Push control towards callee:* In the current communication paradigm, the caller controls how to reach the callee. This is at loggerheads with the concept of personal mobility and personalization. Flexible handling of incoming communication is achieved by pushing control away from the caller to the callee.

2.1. Components of the Architecture

For integration of heterogeneous devices and services, we require (at least) the following functional capabilities.

- Any-to-any data transformation: this would be an important piece that provides device independence. In part, by handling data type transformation in a generic fashion, it would provide *device data type independence*.

- User preference based ubiquitous redirection: this functionality would enable personalized redirection of incoming communication between different end-points of the callee.

- Device or service end-point name mapping and translation: the requirement for this capability can be seen directly from Figure 1. If the user has different end-devices or services with different names, mapping between their identities is necessary for integration. This functionality would provide *device name independence*.

Apart from these components, we have gateways or Access-Points (APs) bridging different access networks to the Internet (Figure 2). They perform the important function of protocol translation (proxying) between the access network protocol and an Internet session protocol (like SIP [30]). They establish and tear-down sessions associated with user-to-user communication or service access. APs use the data-transformation and name-mapping components for device-independent operation.

We have made the crucial design choice to have our architecture be network- rather than edge-centered. That is, we implement the mobility components as services in the core infrastructure, as opposed to colocating them with the access-points at the edges. This is shown in Figure 2. The access-points use the core components for the mobility functionalities. (By core infrastructure, we mean the Internet that is in the middle of all the different access networks

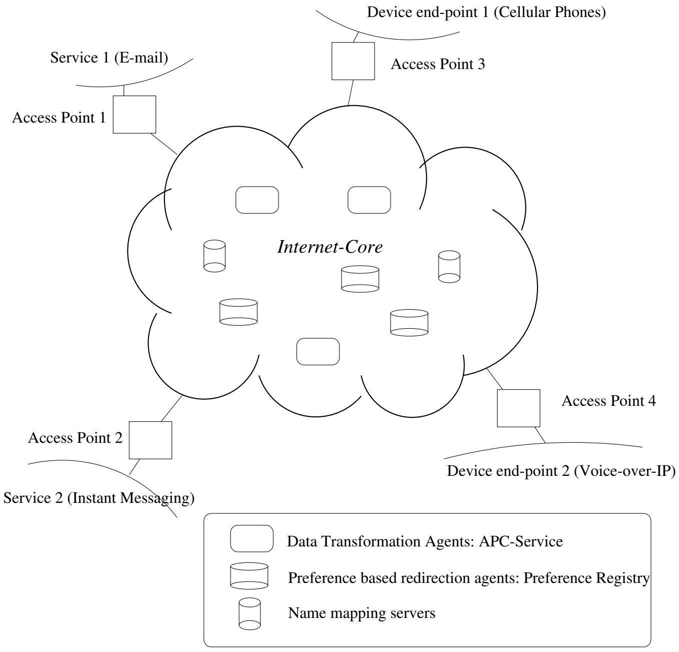

Figure 2. Reusable components in the Internet-core

– this does not necessarily mean the core "backbone" of the Internet).

This is a crucial deviation from the approach taken by other architectures like the Mobile People Architecture (MPA) [28] in which the mobility components are colocated with the gateway functionality at the *edge* of two networks. It is also different from today's integration of the GSM cellular network with the PSTN – this is done through an Inter-Working Function (IWF) at the *edge* of the two networks [24].

This design choice is in accordance with our goals. Placement of functionality in the core simplifies extensibility: reuse of these components simplifies development and deployment of new access points. Components in the core can be shared by multiple access-points. They can be incrementally provisioned to accommodate a growing user community. And their implementation can also leverage scalable cluster service platforms [6, 8].

In the following subsections, we discuss the architectural components in turn.

2.2. Data Transformation Service: Automatic Path Creation

In mobility scenarios involving heterogenous devices or services, some form of data transformation is required. In accordance with our design principle (#1), we separate this functionality into an independent component that can be reused by all mobility features. This component is the *data transformation service*.

The data transformation component leverages the powerful concept of *Automatic Path Creation (APC)* from the Ninja project [11]. In APC, an *operator* is a generic data transformer (e.g., an audio codec), and a *path* is a series of operators strung together [20]. This is depicted in Figure 3. The path shown consists of two operators (the bold lines) – it converts a given piece of text to PCM audio.

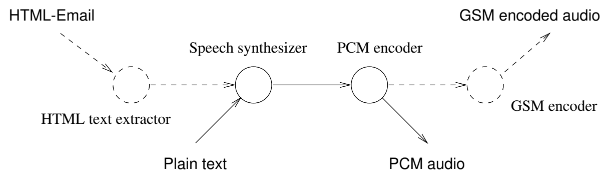

Figure 3. Illustration of a Path

Since this component uses APC, we also call it the *APC-Service*. The term "automatic" refers to the fact that we only need to specify the input and output data formats (and input source and output sink) to the APC-Service, it then strings together the operators necessary for the transformation "automatically".

This provides a highly extensible model. For instance, in Figure 3, suppose one wishes to add support for GSM audio as well, all that needs to be done is to add an operator (see the dotted lines on the right). Similarly, suppose we want to add support for conversion of HTML e-mail messages, we would just have to add the appropriate operator (see the dotted lines on the left).

A generic data transformation component is absent in today's communication infrastructure. Transformations, if any, happen at the edges (e.g., at the Inter-Working Function between the GSM and PSTN networks [24], or at each service provider deploying integration services) and are not generic enough or reusable for other services.

2.3. Preference Registry for Redirection

The second functionality common to personal mobility scenarios is redirection. Redirection is often highly user-specific (refer to Figure 1). This functionality is implemented within the Preference Registry (PR) component. The PR stores and processes user preference profiles and acts as the redirection agent across heterogenous devices. It realizes the "control for the callee" design principle (#3).

User's preference specifies the way a particular incoming communication should be handled. It is a function of several factors like the time of the day, caller-id, user location, user state, and so on. The inputs can be classified into per-session inputs (the first two in the previous list) and dynamic inputs (the last two). The output is the preferred end-point of the callee.

We represent user preference as a set of rules. We model it as a script that is processed by the PR, each time with a new set of input values to the script. Projects such as the MPA [28], TOPS [1], and Active Messenger [22] have used similar ways of representing user profiles and collecting the input parameters. We view these as complementary to this aspect of our system. Figure 4 shows an example of a user preference script.

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

Figure 4. A Simple Preference Script

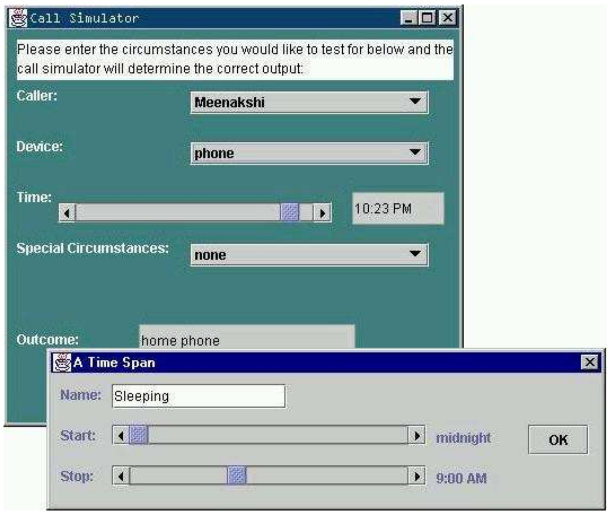

Figure 5. A screen-shot of the GUI for specifying user preferences

The User Interface issue is an orthogonal, albeit important one. Building an interface to let the user specify preferences based on simple parameters like time-of-day, and caller-id is straightforward. It is more complicated to specify rules such as: forward faxes coming into my office as e-mail, if I have not been in the office for 2 days. We have built a GUI for specifying simple preference scripts – a screen-shot of this is shown in Figure 5. We are in the process of perfecting this GUI to perform usage studies in the future.

Just as with format translation, redirection is done mostly at the edges today: the user specifies the number to which incoming calls to a particular end-point should be forwarded. While this is a useful functionality supported by session setup protocols like SIP [30], redirection across devices for personal mobility should really be done at a central point. Placing this functionality at the edges means that it has to be replicated each time there is a new end-point (i.e., the user has to specify redirection for each of the end-points). Centralizing redirection in the Preference Registry is in accordance with our goals of extensibility to new end-points.

2.4. Naming service

The Naming service is a component required for handling heterogeneous devices and services that have different name spaces. It allows us to map between the different user end-point identities in these name spaces. We define a unique-id for a user which is equivalent to MPA's PoID [28] or UMTS's UPT number [26].

The Naming service maps between the identities of the user's different end-devices, or communication services. All these identities map to her unique-id – this allows lookup of further information indexed on the user's unique-id. For instance, the location of the user's preference registry is determined as a mapping from the unique-id.

The Naming service is used as the bootstrap mechanism for locating a user or a service end-point. Hence it needs to be globally distributed and scalable. We consider two options to achieve this. The Domain Name System [23] is a globally distributed mapping service in use in the Internet today. One possibility is to use the DNS for the mappings we want. Name spaces like cell-phone numbers could be converted to the dotted string notation for making them DNS entries. And a special type of DNS record could be used for the entries we are interested in.

A second possibility is to use a hierarchical tree separate from DNS, but use similar mechanisms for distributions and scaling. The first choice has the advantage of using the existing DNS infrastructure; but might end up overloading it. The second choice allows the flexibility of having separate caching and ownership semantics. This is important since the use of our naming entries is likely to be very different from today's DNS. We are continuing to explore the two choices, but have taken the second approach since it was easy for implementation in our testbed.

In either case, extending the name hierarchy is easy. Addition of a new name-space involves the addition of a subtree to the name hierarchy. Figure 6 shows an example where the name hierarchy has been extended to include a new paging service that has a new name space for its pager end-point identities.

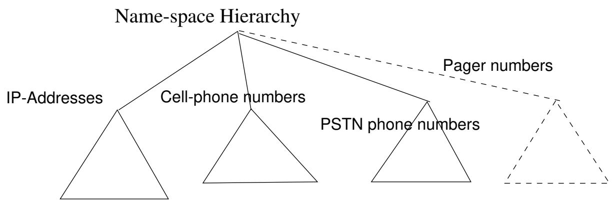

Figure 6. Extending the name-space: an example

2.5. Access Points

An *Access Point (AP)* provides gateway functionality for an access network. It exports a generic session setup interface to the Internet core. The common session setup interface provides a level of indirection that is key to achieving network independence (design principle #2).

An AP could interface to a service or to devices in an access network. For instance, we could have an AP to interface with an e-mail store, and another at the Mobile Switching Center (MSC) [27] of a GSM network to interface with cellular-phones. An AP acts as a *client* when it establishes outgoing (i.e., into the Internet core) sessions, and as a *server* when it accepts incoming sessions.

2.6. Putting it all together: An Example

Figure 7 shows a simple personal mobility scenario using the architectural components introduced above. It shows how personalized redirection would work across heterogeneous device-types. The caller dials a number from the cellular network. The AP at the edge (e.g., at the MSC or the Base-Station) intercepts this (step 1) and gets the unique-id and the location of the preference registry of the callee using the distributed Naming service (step 2). It then gets the current preferred end-point of the callee from his PR (step 3) and establishes a call session through another AP to the callee (steps 4-7). An APC service instance in the infrastructure is used for any data transformation (step 8). (We view the problem of locating an appropriate service instance as an important but orthogonal problem).

3. Extensibility in the Universal Inbox

In this section, we present the extensibility features of the Universal Inbox by describing our experience with building mobility features in the framework presented above. It is important to note that the individual functionalities we describe here are not novel by themselves. But the way they are built by (re)using the separate components is what is unique. In Section 3.1, we list the functionalities we have

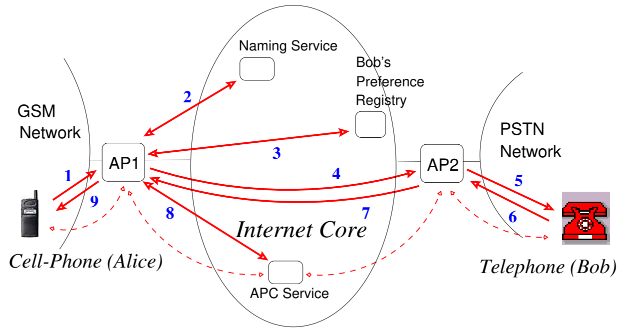

Figure 7. Putting it all together: An Example

implemented incrementally and describe the different extensions we have made to the Universal Inbox. Then in Section 3.2 we discuss these extensibility features and also some of the caveats.

3.1. Implementation Experience

Figure 8 and Table 1 show the step-wise addition of end-points and features to the Universal-Inbox. Specifically, they show the Access-Points, and operators at the APC service that were added. During each step, additional preference profiles and naming entries were also created. However, these are not shown since these are only configuration changes and not functionality additions. The "+" at the beginning of each entry in Table 1 denotes that the entry represents an *addition* to the set of end-points, operators, and features. (All of the APs in the table have been implemented, except the PSTN AP – row #7).

To start with (row #1 of Table 1), we have two kinds of end-points in implementation: (a) GSM cell-phones, and (b) Desktop voice-over-IP end-points in the form of the Visual Audio Tool (VAT) [19]. Each kind of end-point is reachable through an Access Point. The GSM AP interfaced to the cell-phones through a Base-Station and a BSC/MSC simulator; the details of the interface are not relevant here. The VAT end-points use the GSM codec and we have a single data type in our system: GSM audio.

The name hierarchy has entries for IP-addresses and cell-phone numbers. Personalized redirection across the heterogeneous end-points is possible through the use of the PR and Naming service components. The example in Section 2.6 showed how this would work.

Extension of the redirection functionality to include voice-mail involves the addition of an appropriate AP (row #2). This AP also allows a user to access her voice-mail through either of the earlier two end-points. Note that the development and deployment of this voice-mail AP is independent of the previously existing APs.

The first text-based end-point we integrate is e-mail (row #3). For this, we implement a client Access Point that resides at the user's e-mail store and establishes outgoing sessions for reading out e-mail to the user's preferred end-point. For each e-mail received in the user's inbox, the Access Point checks the user's Preference Registry to see if (a copy of) the e-mail should be redirected to another end-point (e.g., the user's cell-phone).

The support required for this at the APC Service is three operators for text to speech conversion: (a) text to sun-audio (based on festival [5]), (b) sun-audio to PCM (based on the sox Unix program), and (c) PCM to GSM (based on the toast GSM codec [3]). With this in place, e-mails can now be redirected to *all* previous end-points.

Next, we do a similar integration with an instant messaging service, by implementing an AP for the same (row #4). The AP interfaced with the Sanctio instant messaging service that was developed as part of the Ninja project [11]. We can now reuse all of the APC functionality added in the previous step. The functionalities available with e-mail earlier are now readily available with instant messaging as well.

We now enable access to two services in turn. The first is the Jukebox service (row #5) – which was also developed independently as part of the Ninja project [11]; the service plays streaming MPEG3 encoded music to the user's desktop. We add an AP to proxy for this service, and also add an operator at the APC service (MPEG3 to PCM converter, based on the mpg123 unix program). We reuse the PCM to GSM operator added earlier, and access to this service is now enabled from the device end-points: cell-phones and VoIP desktops.

The second service we enable access to is the MediaManager service (row #6). This was also developed independently as a separate project. It is a service that sits in front of the user's e-mail store, and is capable of doing intelligent processing (such as summarizing) on the e-mail. Enabling access to this involved building the AP to proxy for it. The MediaManager is capable of outputting several audio formats including the GSM format. Hence no additional operators are required at the APC service.

Finally, we consider how we can add PSTN telephone end-points to the system (row #7). This capability has not been fully implemented yet. We have a H.323 gateway [13] interfacing with the PSTN network. We need to add an AP in front of this gateway. This gateway supports only the G.723 audio format. To interoperate with GSM-based end-points, we need to add the three operators shown in the table. With this done, all of the previous functionalities: redirection, screening, and service access that were possible with the earlier end-points, will now be possible with PSTN telephones as well.

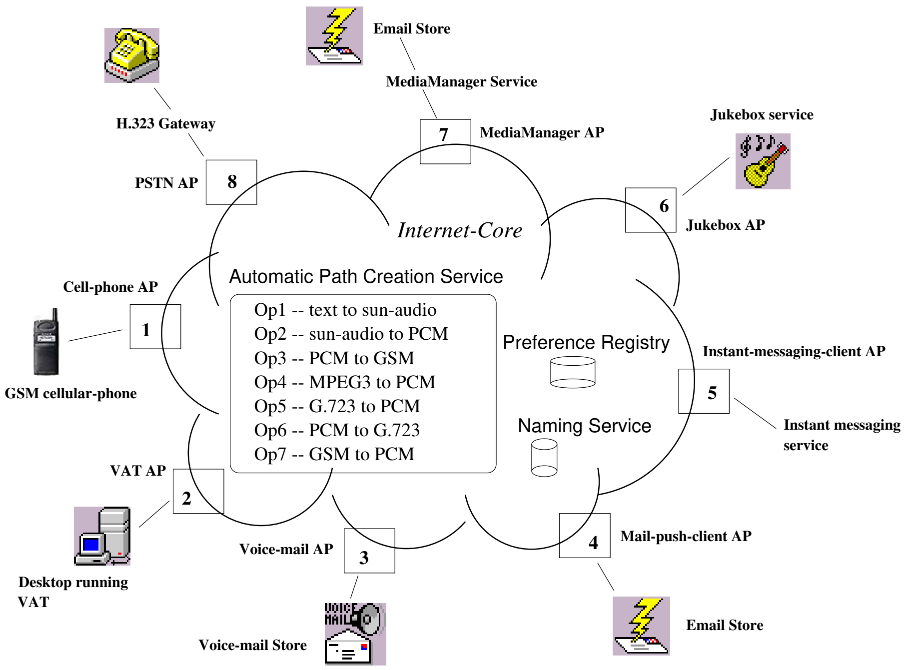

Figure 8. Step-wise additions to the Universal-Inbox (refer Table 1)

3.2. Discussion of Extensibility Features

In general, extending the system to new devices or services involves the addition of an Access Point. An Access Point essentially provides the glue for integration. It has a device- or service-specific part, and it has a generic part. The generic part is the Internet session establishment protocol, such as SIP [30]. In implementation, we have used a simple Java RMI based session initiation and termination protocol, since this is not the focus of our work.

The device- or service- specific part of the AP could be quite complicated. For instance, developing this for the GSM AP required understanding the GSM protocol stack and was a considerable effort of about nine person-months. However, APs to simple services can be developed easily. The interfaces to the Jukebox service and the MediaManager service are quite simple – JavaRMI calls to retrieve songs and messages respectively. APs to these services are only about 700 lines of Java code each – one could consider adding more frills to these APs though.

While extending the system, in some cases, when there are new data formats involved, operators have to be added to an APC service instance. Here again, an operator may not be easy to implement by any means. For example, a text to speech conversion software is non-trivial. But the key is that such functionalities can be reused – not only in implementation, but also in *deployment*. That is, a third party APC service in the Internet-core can be used for the appropriate conversions. This is richer than the reuse of functionality that is possible with other architectures like the MPA [28]. An MPA Personal-Proxy (PP) can reuse the "conversion drivers" for different mobility features. But such reuse is not possible across two instances of the PP.

The deployment of new services and integration of new devices is helped by the fact that the access points are *independent*. They can be developed and deployed at different points in the infrastructure, by different parties.

The extensibility features of our architecture are enabled by the separation and reuse of components. While this is a rich feature, there could be drawbacks in some cases. If the interfaces between the components are defined over a network, they cannot be as rich as they can be when components are tightly coupled. Consider for example, the case of the MPA where the Personal Proxy encapsulates both the Preference Registry functionality as well as APC functionality. It can implement services like "if the e-mail has the words *free coffee*, and I'm at my office, beep my pager so that I don't miss out". This task would involve close interaction between the data transformation component and the user preference processing component and cannot be done easily in our case where the two could be separated over a network. Thus the extensibility and reuse gained by the separation means the loss of some flexibility.

Separation of components could also mean additional latency. In Figure 7, the Naming service, PR, and APC have to be accessed separately. This could add to the call setup latency. However, this can be addressed by placing these components so that they are not far apart in the network.

		Device/Service AP	Operators in APC	Personal/Service mobility features
1		Cell-phones (#1) & Voice-over-IP (#2)	(none)	Call redirection/screening based on time-of-day & caller-id
2	(+)	Voice-mail (#3)	(none)	Call redirection to voice-mail also possible Voice-mail access from cell-phone/VoIP end-points
3	(+)	Mail-push-client (#4)	Op1 (text to sun-audio) Op2 (sun-audio to PCM) Op3 (PCM to GSM)	Email redirection to cell-phone/VoIP/Voice-mail
4	(+)	Instant-message-client (#5)	(none)	Instant message redirection to cell-phone/VoIP/Voice-mail
5	(+)	Jukebox-service (#6)	Op4 (MPEG3 to PCM)	Jukebox access from cell-phone/VoIP
6	(+)	MediaManager-service (#7)	(none)	MediaManager access from cell-phone/VoIP
7	(+)	PSTN end-points (#8)	Op5 (G.723 to PCM) Op6 (PCM to G.723) Op7 (GSM to PCM)	Call redirection to PSTN, E-mail redirection to PSTN Instant-message redirection to PSTN Jukebox & MediaManager access from PSTN

Table 1. Step-wise additions to the Universal-Inbox (refer Figure 8)

For instance, if a telephone service provider deploys an access point at a switch, the service provider could also deploy (or use) an APC service near the AP to reduce the call setup latency.

4. Scalability Analysis

In our design, the three architectural components are implemented and deployed as shared infrastructure services. There are scaling and provisioning concerns that have to be addressed for such shared services. This section presents the results of stress-testing our implementation for scalability. The numbers in this section give an idea of the compute resources required to support a given size user-community. We also provide latency measurements; however, they do not include wide-area network latencies, since our current testbed does not span the wide-area.

Among the three components, the naming service is light-weight by nature, since it only involves name-lookups, and can benefit from optimizations like caching. Hence we focus on the performance of the other two components. We now briefly describe the main features of our implementation before presenting the results of the performance experiments.

4.1. Relevant Implementation Details

The APC Service and the Preference Registry components are Java-based implementations. In accordance with our goals of scalability, the APC Service is structured for a cluster-based implementation. It has a front-end and several back-ends; each back-end is capable of running operators. The front-end decides which operators to run and where. The operators themselves are all in C, since they perform the bulk of transformations, and have Java wrappers at the back-end nodes.

The PR also has a front-end and a back-end. The back-end implements the storage of the user preference scripts, and the front-end processes them. The back-end uses a distributed, persistent, cluster-based storage mechanism [7]. The scripts are represented in TCL and the front-end uses a Jacl interpreter to process them [16].

For all the tests, we used 500Mhz Pentium-III 2-way multiprocessor machines for the servers (i.e., the functional components) and a 400MHz Pentium-II machine for the client. The servers had 256MB of main memory and 512KB of processor-cache in each of the processors. The servers were in a cluster, and the client was separated from them by 6 hops within an in-building LAN (approx. 1ms round-trip time). All machines were running Linux-2.2. All the Java programs used IBM's JDK v1.1.8.

4.2. The APC Service

At the time of this writing, the APC service has gone through one round of performance optimization. Our original implementation modeled operators as unix processes; one such process would run for each path in a back-end node. This did not scale well due to the overhead of a process per operator per path.

In the next round of implementation, we have modeled operators as being sharable by multiple paths. A single operator handles multiple paths by reading from and writing to different sockets. We show the performance of this optimized implementation below. We have implemented the toast (PCM-to-GSM) and untoast (GSM-to-PCM) operators under this optimized model. We have not yet implemented the other operators in this model. For comparison, we also implemented a *null* operator that simply copies its input to output (we use PCM data rate for this operator).

A path persists for the duration of a session. Sessions may last for varying periods of time. One can expect a

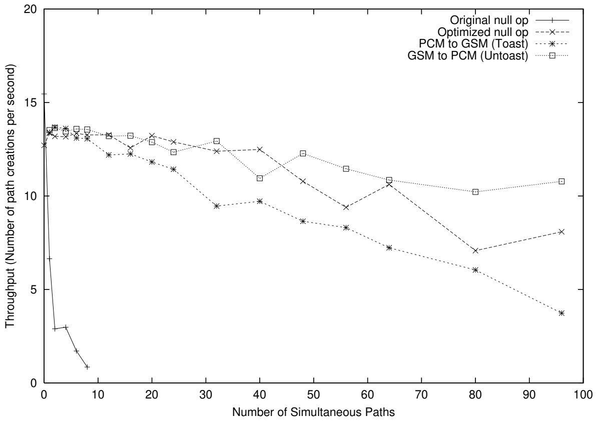

Figure 9. Path creation throughput

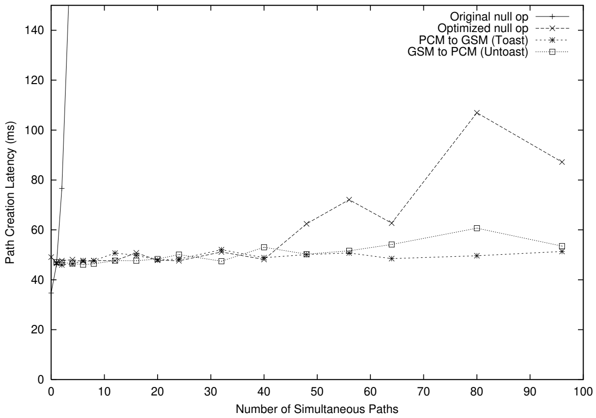

Figure 10. Path creation latency

certain number of paths in service at any given time. This number averaged over time is a measure of the load at the APC service. For measurement control, we fix this metric of load for the duration of an experiment (i.e., we create a fixed number of paths at the beginning) and vary it across different runs. With this fixed load at the server, a client repeatedly creates and tears-down further paths. All the pre-existing paths had data streaming through them. The data rate is one audio frame per 20ms; a frame is 160 bytes for 8kHz PCM data and 33 bytes for GSM data.

Under this setup, we measure two parameters: path creation throughput and latency. Figures 9 and 10 show these two parameters as a function of the load (i.e., the number of simultaneous paths created at the beginning of the experiment). We use separate front-end and back-end machines for these experiments. The graphs also show the corresponding measurements for the optimized and unoptimized *null* operators.

The measurements show that an average load of at least 60-70 simultaneous calls can be supported before the throughput drops to half its value under zero-load – for the toast operator. This number is even higher for the untoast operator. The unoptimized operator scales very poorly since it uses a unix process for each path.

The performance of the toast operator goes down more rapidly than untoast: GSM encoding is more heavy-weight than decoding. The performance for the (optimized) null operator is actually worse than that of the untoast operator. A possible reason for this is that the null operator used a higher outgoing data rate than the untoast operator (the null operator used PCM data rate on both sides while the untoast operator used a lower data rate GSM audio for the incoming data). This would mean that the operations are not CPU intensive, but I/O intensive. We have actually confirmed this and the null operator with the GSM data rate does scale better. We had a throughput of 11 path creations per second at a load of 80 simultaneous paths. We have not included this in the graph for the sake of clarity.

Calculation of Scaling

We now map the numbers presented above to the number of users that can be supported. We do this in the context of two-way telephone calls since there are extensive statistics available for this.

Suppose the average call arrival rate is R, and suppose an average call lasts for t, the average number of calls at a given time is $L = t \times R$ at steady state. The maximum throughput of the system (rate of path creation) should be greater than the value of R for the system to keep functioning steadily.

From the studies in [21], at the busy hour, $R = 2.8\ calls/hour/user \times N$ (N is the number of users in the system) and $t = 2.6\ minutes$. From Figure 9, the throughput is $7.2\ paths/sec$ when $L = 64$. $R = L/t = 0.44$ calls/second. The throughput is indeed greater than the arrival rate R.

The number of users is calculated by $N = R/(2.8\ calls/hour) = 571$. This means that a user community of over 500 can be easily supported by our two-node APC service. This is encouraging given that these operators are reasonably complicated. The telephone network employs special, expensive hardware equipment for doing these transformations (see TRAU in [25]). We expect even better scaling with further optimizations.

Latency through a path

Finally, we present the latency experienced by a data stream through a path, after path creation. Table 2 shows these numbers for the optimized toast and untoast operators and for the original unoptimized null operator for various path lengths. The numbers in parentheses give the standard-error for 200 observations; some of the large variations are not surprising given that the round-trip between the client and the server itself was 1ms. Even in the case of multiple operator paths, all the operators were running on the same back-end machine. These latencies are quite low compared to typical network latencies; however, further optimizations are certainly possible.

Operator	Latency through path
Toast op	9.5ms (0.19ms)
Untoast op	11.8ms (0.46ms)
1-null-op path	3.8ms (0.48ms)
2-null-ops path	5.3ms (0.90ms)
3-null-ops path	4.5ms (0.09ms)

Table 2. Latency through a path

Throughput	55.3 requests/sec
Latency breakdown	
Backend retrieval	8.9ms (0.51ms)
Script Interpretation	17.7ms (1.32ms)
JavaRMI	9.4ms (2.3ms)
Total	36.0ms (1.79ms)

Table 3. PR throughput and latency

4.3. The Preference Registry

We now present the performance numbers for the preference registry. In these experiments, the PR was pre-loaded with preference scripts (dummy ones) for 5,000 different users. Each script was 50 lines of TCL. Although the back-end store can be distributed over multiple nodes, the experiments use only a single node. This single node also runs the front-end.

We had two different client threads to repeatedly access the preference registry. Table 3 shows the throughput and latency of preference registry access. The table also gives the breakdown of the latency. The RMI latency was computed by subtracting the sum of the latencies at the server from the total latency at the client.

Calculation of Scaling

If we consider the call arrival rate of 2.8/hour/user (from [21]), there would be $N \times 2.8/3,600$ call setups per second and the corresponding same rate of PR requests per second. Using the value of 55.3 lookups/sec from Table 3, we estimate that a single PR can support $N = 71,100$ users. Given that we loaded the PR with only 5,000 user preference scripts, it would be safe to say that at least 5,000 users can be supported by a single machine PR.

4.4. Additions to Session Setup latency

The Naming Service, PR and the APC-Service have to be accessed separately during session setup (refer to Figure 7). From Table 3 and Figure 9, we see that the accesses to the PR and the APC-Service add approximately 36ms and 50ms respectively to session setup. The Naming Service would also add latency, but extensive caching can help here. At the APC-Service, the breakdown of latency was: 5.5ms for JavaRMI, 19.2ms for communication between the front-end and the back-end (JavaRMI again), and 25.4ms for instantiating the path at the back-end node. These numbers are for the toast operator when there was a load of 48 simultaneous paths.

Although these latencies may be acceptable in some scenarios (like asynchronous e-mail to voice conversion), they are quite high for regular call setup. We expect to bring these numbers down in the next round of implementation.

5. Related Work

We have compared our architecture to existing ones in several contexts in the previous sections. This section summarizes these comparisons.

There has been a lot of recent commercial interest in service integration. There are several efforts that provide partial integration of communication services. These include services like e-mail/voice-mail integration [12, 15], e-mail/fax integration [10, 14], etc. Some provide only "name" integration, and no "type" integration. None provide true any-to-any, extensible integration or personalized redirection across *all* devices.

The Active Messenger project [22] provides elaborate mechanisms for delivery of messages to the user at any end-point. It concentrates on this functionality – which is an important, but orthogonal problem to what we focus on. We present an infrastructure component based approach for extensible integration on top which functionalities like the Active Messenger agent can be built.

There have also been recent research efforts with respect to Personal Mobility. The Mobile People Architecture (MPA) [28] is an architecture based on a Personal Proxy (PP) for achieving person-level routing; the PP is responsible for handling communication on the user's behalf. Telephony Over Packet networkS (TOPS) [1] is an architecture that provides redirection of incoming communication using a Terminal-Tracking-Server.

The clear identification and separation of the functional components in design, implementation, and most importantly, deployment, is unique to our architecture. The Universal Inbox architecture maps these components to a set of reusable network components. As we argued in Section 3, this has immense advantages in terms of extensibility; especially with respect to the data transformation component.

MPA and TOPS identify components for user preference management and name translation. MPA also identifies a component for data transformation (the conversion drivers).

However, the components are tightly coupled and are not realized as reusable network services. This restricts the extensibility and scalability of these architectures. For instance, in the case of the MPA, the PP has to know about all possible data formats (i.e., have the appropriate "Conversion Drivers"). If there is a new service with a new dataformat that needs to be integrated with the system, the PP's of all users have to be changed to accommodate this. With the Universal Inbox, since the data transformation component is an independent third-party service, only this has to change. All users from all devices can now use the new service (Section 3 provided specific examples to illustrate this).

Furthermore, the Universal Inbox uses the notion of a *path* of data transformation [20], which provides an extensible data transformation mechanism. This gives us additional extensibility features.

Finally, UMTS [29] is an ETSI standardization effort that includes third generation PCS services; personal mobility across device end-points is one of its features. It makes use of Intelligent Network components for realizing its functionality [2, 4]. Despite being a highly scalable architecture based on the SS7 network, it does not identify explicit components for preference based redirection or data transformation. Furthermore, its SS7 based architecture has implications on the high cost of entry to adding novel functionality [18].

6. Conclusions and Future Directions

Providing personalized integration of heterogeneous user devices is of crucial importance for communication management. There has been a lot of recent commercial as well as academic interest in integrating user communication end-points. In this paper, we take a close look at what it takes to provide any-to-any integration in an extensible and scalable fashion in an integrated network. We identify three crucial components for such an integration, and present an architecture mapping these to reusable infrastructure services.

The key strength of our architecture is the **extensibility** of the system. The network-centric approach makes the system easy to extend to new end-points and services; the reusable components in the core provide the "glue" to integrate with *all* existing end-points. Ease of extensibility also comes from the flexible *path* model upon which the data transformation component is built. Ease of deployment in the architecture comes from the fact that service-specific access points can be deployed independently; the user's e-mail service provider can build an access point independent of their pager service provider and the user can still manage these two services in an integrated fashion.

Personalization is provided by a central redirection agent rather than at the edges; this means that it is easy to personalize the management of *all* of a user's devices. Our initial performance experiments show that it is indeed possible to **scale** the infrastructure components to a large user community, which would be required when these components are implemented as reusable services.

Our implementation of the mobility features has given us experience with building and extending services using the component building blocks. A variety of issues remain for future work.

We have assumed a cluster-based implementation of the APC service in which data paths are within the cluster. However, there may be cases when this is not possible and a path has to be stretched across the wide-area. There are interesting issues of path-control and monitoring in such a scenario. Also, infrastructure services that have multiple instances can take advantage of the Internet topology to be strategically placed such that a service instance is always available for the client of the service. We are currently exploring this issue. Finally, we have not addressed issues related to secure billing for services in the integrated network, which are crucial for a real deployment of the system.

Acknowledgements

We wish to thank all the members of the ICEBERG project for the healthy discussions that helped in refining the design. We are grateful to Tina Wong, Drew Roselli, Sanjay Rao, and the anonymous reviewers for their comments on earlier revisions of the paper. Rahul Biswas implemented the Preference Manager GUI. Emre Kiciman, Barbara Hohlt, and Z. Morley Mao worked on the implementation of the APC service. This work is part of the ICEBERG project at U.C.Berkeley supported by Ericsson, Sweden.

References

[1] N. Anerousis and et.al. 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] E. Buitenwerf and et.al. UMTS: Fixed Network Issues and Design Options. *IEEE Personal Communications Magazine*, February 1995.

[3] J. Degener and C. Bormann. GSM Codec Library, University TU-Berlin, FB-Informatik. http://kbs.cs.tu-berlin.de/ jutta/toast.html.

[4] N. Faggion and C. T. Hua. Personal Communications Services Through the Evolution of Fixed and Mobile Communications and the Intelligent Network Concept. *IEEE Network*, Jul/Aug 1998.

[5] Festival. The Festival Speech Synthesis System. http://www.cstr.ed.ac.uk/projects/festival.html.

[6] A. Fox and et.al. Scalable Network Services. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP-16)*, St. Malo, France, Oct 1997.

[7] S. Gribble. Simplifying Cluster-Based Internet Service Construction with Scalable Distributed Data Structures. *PhD Candidacy Qualifying Exam, U.C.Berkeley*, April 1999.

[8] S. D. Gribble and et.al. The MultiSpace: an Evolutionary Platform for Infrastructural Services. In *Usenix Annual Technical Conference*, June 1999. Monterey, CA.

[9] J. Homa and S. Harris. Intelligent Network Requirements for Personal Communication Services. *IEEE Communications Magazine*, February 1992.

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

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

[12] http://planetarymotion.com/. *Planetary Motion's CoolMail Service: Email by Computer or Phone*.

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

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

[15] http://www.onebox.com/. *Onebox: Voicemail, Email, and Fax*.

[16] http://www.scriptics.com/java. *TCL Java Integration*.

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

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

[19] V. Jacobson and S. McCanne. VAT Mbone Audio Conferencing Software. ftp://ftp.ee.lbl.gov/conferencing/vat.

[20] A. D. Joseph, B. Hohlt, R. H. Katz, and E. Kiciman. System Support for Multimodal Information Access and Device Control. Work in Progress, Workshop on Mobile Computing Systems and Applications (WMCSA), 1999.

[21] C. N. Lo, R. S. Wolff, and R. C. Bernhardt. An Estimate of Network Database Transaction Volume to Support Universal Personal Communications Services. In *First International Conference on Universal Personal Communications (ICUPC '92)*, 92.

[22] S. J. Marti. Active Messenger: Email Filtering and Mobile Delivery. Master's thesis, MIT Media Laboratory, September 1999.

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

[24] M. Mouly and M. B. Pautet. *The GSM System for Mobile Communications*, chapter 2, pages 102,133. Cell & Sys, 1992.

[25] M. Mouly and M. B. Pautet. *The GSM System for Mobile Communications*, chapter 3, pages 150–153. Cell & Sys, 1992.

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

[27] M. Rahnema. Overview of The GSM System and Protocol Architecture. *IEEE Communications Magazine*, April 1993.

[28] M. Roussopoulos and et.al. Personal-level Routing in the Mobile People Architecture. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, Oct 1999.

[29] A. Samukic. UMTS Universal Mobile Telecommunications System: Development of Standards for the Third Generation. In *IEEE Transactions on Vehicular Technology*, volume 47, Nov 1998.

[30] H. Schulzrinne. A Comprehensive Multimedia Control Architecture for the Internet. In *Proceedings of International Workshop on Network and Operating System Support for Digital Audio and Video*, May 1997.

[31] M. Zaid. Personal Mobility in PCS. *IEEE Personal Communications*, Fourth Quarter 1994.