The Thinnest Of Clients: Controlling It All Via Cellphone

Frank Stajano[*†]
fstajano@orl.co.uk

Alan Jones[*]
ajones@orl.co.uk

Abstract

The *thin client* paradigm aims to give users access to central resources through inexpensive and easily deployed computing systems. But, however "thin" the client hardware, mobile users in the field still have the burden of carrying it. To alleviate this problem, we decided to adopt as our client a piece of hardware that many mobile users already carry with them anyway: the cellphone.

This paper presents our experience in researching, implementing, deploying and using a system whereby users, wherever they are, can query and control their personalised computing resources and services by typing short messages on the keypad of their cellphone. Our system has been deployed and in use for over a year and has given us valuable insights on how to design and build a personal information service.

I. Motivation

A. Caravan or hotel?

Most of the practical offerings that attempt to solve the problems of the mobile user involve carrying around a smaller and somewhat feature-impaired version of the user's deskbound computing equipment. From laptops and notebooks to PDAs and palmtops, manufacturers have over the years explored most of the spectrum that spans the seemingly inevitable trade-off between functionality and portability.

Another way to attack the problem, which we have been pursuing at ORL for several years now [1, 2, 3] makes it possible to avoid the trade-off and thus benefit from great functionality and portability at the same time: it's what we like to call the *"Would you rather stay in a caravan or in a hotel?"* alternative. It's the realisation that, instead of bringing with you a portable computer with a scaled-down version of your environment, like a caravanner, you can instead travel lightly and use the comfortable facilities available at destination, like a hotel user, making your home environment accessible on the remote computer by means of special software such as our VNC [3] system.

B. What if no computers around?

The underlying assumption of this successful approach has been that it will be possible for the mobile user to temporarily log on to a locally available computer with suitable network connectivity. What, however, if this assumption no longer holds? It is not totally inconceivable for the user to travel to a location not served by computers or in which the local computing facilites won't be easily available to a visitor. It is also possible for the user to wish to access computing resources (particularly for simple tasks such as looking up a timetable) while on a train or waiting in a bus queue.

The traditional answer to this need has been to provide special miniaturised computers, with varying degrees of data connectivity, sufficiently small that a user could actually be persuaded to carry them around all the time: we refer to palmtops, of which we have personally been keen users since 1991. We have to admit, though, to our fondness for computer gadgets: from data-linked wristwatch to palmtop computer and from bicycle-mounted GPS receiver to pocket web browser, each of us usually carries more electronic gizmos than the regular human being has pockets. We thus reluctantly recognise that the palmtop solution is still mostly dedicated to a niche market: if we were to propose a solution based on carrying a new electronic device, however useful, we would probably get an enthusiastic response from the closed community of our fellow gadget freaks, but we wouldn't be able to persuade the wider community to adopt it.

The shocking revelation is, however, that there *is* one electronic gadget that regular human beings carry around spontaneously everywhere they go—the cellphone. We noticed that, if we could use this ubiquitous representative of the digital world as a gateway into the global cloud of computers and communications, we would then be in a position to offer a practical solution to a majority of mobile users. This strategy is a natural extension of the thin client *"caravan or hotel"* paradigm of making use of existing available facilities instead of forcing the user to carry around new equipment; the subtle difference here is that the user *is* actually having to carry an accessory around, but it's something that the user was willing to carry anyway, regardless of our intervention.

C. The SMS server

It would be foolish to claim that a system whose user interface consists of the 12-key alphanumeric keypad and tiny display of a standard cellphone could possibly match the capabilities of a real computer system—be it a fixed desktop system, a virtually remoted session, a laptop or even a humble palmtop. Our system is certainly not a complete solution for the mobile user who needs to access computing resources. It is, however, a convenient addition to the array of solutions available to that user. It adds some extra facilities that make particular sense to the user on the move, some of which aren't normally accessible in a convenient form even from the most complete desktop systems. The ubiquity of the cellphone (any manufacturer's model will work as long as it supports the GSM standard's Short Message Service) means that our system is always available anywhere and at any time, without the user having to remember to carry any special supplementary equipment.

As a matter of illustration, here are some of the possibilities opened up by our system; all the scenarios described, unless

[*]ORL, the Olivetti and Oracle Research Laboratory, 24a Trumpington Street, Cambridge CB2 1QA, UK.

[†]University of Cambridge Computer Laboratory, New Museums Site, Cambridge CB2 3QG, UK.

otherwise noted, have actually been implemented at our laboratory and have been available for daily use since approximately mid-1997 to members of our staff.

1. The user can, while queueing for a taxi or sitting in a pub, consult any of a multitude of information servers giving train time tables, weather forecast, traffic conditions, stock market quotes, currency exchange rates and so forth.

2. One of our colleagues, having released some very successful free software on the company's web page, has written an ego-stroking personal handler telling him the number of times his package has been downloaded so far.

3. Another service that we thought of a long time ago, and that we shall deploy as soon as the sensing technology from a sibling project is ready, will allow the car user heading for the office to query the availability of parking spaces at the front and back of our premises, thus avoiding a trial-and-error process that is made time-consuming by morning traffic.

4. Instead of polling for information, the user can also configure the system so as to be notified asynchronously when an event of interest occurs: the cellphone will then discretely beep with an explanatory message when the share price goes outside a predefined band, when the download counter exceeds 10000 or when a news item mentioning your company appears on the teletext.

5. The same "filtering and pushing" concept can be profitably applied to email: a warning will be sent out when, for example, mail from the boss (or Significant Other) is received. This indeed highlights the usefulness of our extremely thin client as a *complement* to the other connectivity options available to the mobile user: most of the network-based activities that one typically performs on one's laptop while on the move consist of *checking* for some event such as the arrival of important email. By asking the SMS server to monitor such events, or by requesting a condensed listing of the headers of the messages so far received, the user can be notified of whether and when it is really worthwhile to connect back to base over that expensive intercontinental modem connection.

6. Speaking of email, the user can of course send a brief email message directly from the cellphone; but there's more: the gateway is bidirectional and the message's headers are composed in such a way that, if the recipient replies via email, the system will forward the reply to the originating phone.

7. In conjunction with our Active Badge™ system [4], the user from the field can obtain the phone number to which a colleague is nearest and phone the colleague directly without going through reception; conversely, the user can also post an "away" message on the Active Badge system, specifying her whereabouts and cellphone number, to allow other colleagues to contact her.

8. The GPS navigation application running in one of our cars uses the SMS service to exchange information with the server. The location information is made available to the Active Badge service so that colleagues and friends can access it in a familiar way: instead of just in the usual in-building locations such as `Meeting room` or `Reception area`, the user may now also be reported as being `In car near Luton`.

9. Finally, given a computer-controlled "smart home" setup (which we haven't wired in yet), the user living on her own could, before cycling home, send a command from the phone to turn on the central heating system just at the right time for the house to be warm when she's back.

II. How it works

A. Physical structure

At the lowest level, the system relies on the "Short Message Service" (SMS) [5] defined by the European GSM standard for digital cellphones. SMS provides a way to transmit textual messages of up to 160 characters between GSM handsets. Contrary to the conventional voice call service, the short message service does not require end to end connectivity; it is instead a store and forward service, much like email, in which the sender delivers the message to the network without knowledge of the connectivity status of the recipient, leaving to the network the task of buffering the message and delivering it when the recipient is next sighted. In normal use, the sender composes the message on the 12-key alphanumeric keypad of the phone and the receiver reads the message off the unit's tiny alphanumeric display, scrolling where appropriate. The physical limitations of the terminal equipment, imposed by the manufacturers' desire to produce the smallest possible handsets, make the 160 character constraint more than reasonable for normal human-driven interaction, since messages much longer than this would be impractical to both type and read on a typical GSM phone.

The system we built consists of a fixed "SMS server" at our laboratory interacting with GSM handsets carried by members of our staff. The server's hardware is a Linux PC with a permanent internet connection on one side and a PCMCIA connection to a GSM handset on the other. The server acts as a bridge between the "computer and internet" world and the "cellphone" world. The server software, written in Python [6],

runs 24 hours a day and listens for requests from both sides: computer events such as the arrival of mail can trigger phone events such as the sending of a short message, and vice versa.

B. Phone-initiated requests

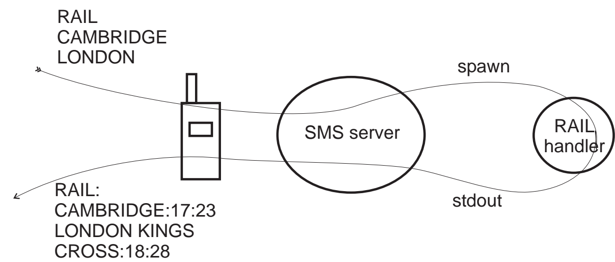
The phone-initiated requests are those in which the user "pulls" information out of the server when she desires. The typical "pull" interaction involves, for the server, one incoming and one outgoing short message. First, the user sends a short message requesting a particular service. The server then performs the service and sends out a response to the originating phone. In the case of a query, such as "tell me the current share price of Coca Cola", sending out the response is actually a fundamental part of the service; whereas in the case of an action, such as "send this email to `friend@company.com`", it is not actually *necessary* for the server to send an SMS reply to the user. It must however be noted that the store and forward nature of SMS transport, with no a priori guarantees about the delivery, gives the user the feeling of sending messages into a black hole: without a response from the other end, the sender can't tell whether the message is being successfully acted upon, whether it is being delayed by traffic in the network or whether it has been ignored due to equipment failure at the server end. For this reason we decided that, as a matter of good user interface, the server should always[1] send a reply to an incoming short message, regardless of the nature of the requested service, just to make the caller aware of the completion of the operation.

1. *"Pull" services: the handlers*

For phone-initiated requests, we modelled the structure and behaviour of the SMS server on that of a web server responding to CGI requests. The server receives a request specifying a particular service and some optional parameters; then, based on the requested service, the server spawns one of a number of external handlers, passing it the parameters supplied by the user; finally, when the handler completes, its output is returned to the user in the short message reply that the server sends to signal completion. This architecture allows us to build a modular system whose range of services can be expanded at any time by simply writing a new handler.

Handlers are separate executables that do not have to be linked to the server and can be written in any language a developer may be familiar with, including shell scripts. The API connecting the server to the handlers is purposefully kept extremely simple. The first word of the short message received by the server identifies one of the available handlers; the remaining words are passed on verbatim to the handler as command line parameters; finally, anything that the handler prints on its standard output while it runs is collated into a string, prefixed by the name of the handler and, after truncation to 160 characters, sent back as the server's response. If execution of the handler exceeds a predefined time limit, the handler is terminated and the server sends out a response that says so.

[1] Actually, to guard against bugs and potential abuses, the server will stop sending replies after certain safeguard quotas are exceeded, as detailed in section III.



1. The *share handler* gives you a current quote for a security listed on any of a number of American and European stock markets. Numerous options allow you to see, among other things, the profit or loss since you bought the shares, with optional currency conversion in case of securities listed on a foreign exchange. Information sources consulted include `www.stockmaster.com` and `www.yahoo.com`.
2. The *weather handler* takes in a city name and gives you a week's worth of daily forecasts with expected weather and minimum and maximum temperature (from `weather.yahoo.com`).
3. The *active badge handler* tells you the location and nearest phone of a named colleague. The "away" subservice lets you post a message to say where you can be reached—this is normally done in advance through a dedicated application running on our computer system, and it is particularly useful to be able to perform this operation when one already is away from the office, having forgotten to register the message before leaving.
4. The *email handler*, as in fact the "away" service we just mentioned, isn't strictly speaking a "pull" service, but still uses the handler architecture. It lets you send a short email message to any user on the internet.
5. The *train handler* takes in the names of two British towns and gives you the time of the next available train from one to the other. It can optionally also show travelling times for times and days other than "right now". Information comes from the `www.railtrack.co.uk` web site.
6. The *currency handler* gives you conversion rates between any two currencies or converts a given amount of any currency into any other, using the service at `www.xe.net`.
7. The *road handler* accepts the name of a major road and searches BBC2's teletext pages for relevant traffic congestion information.

It ought to be apparent that most of these "pull" services follow a common pattern. In a research environment geared towards experimentation such as our laboratory, the ease with which new handlers can be plugged into the server might easily have encouraged a "copy and modify" style of programming in which every new handler is derived by slightly editing the previous one, with maintenance nightmares when one of the old handlers is updated (or fixed) and the newer handlers still contain variants of the old code.

We were careful to avoid this trap by distilling the common behaviour into general purpose classes from which a new handler can inherit. Though the great variety of possible cases still

means that some tweaking may be required, in many circumstances defining a new handler is simply a matter of deriving a subclass describing the source web site and redefining a regular expression meant to match the pages from the site and to extract the relevant fields from it. The rest of the logic, from querying the server with HTTP (potentially filling CGI forms along the way) to filtering the result through the regular expression and massaging the extracted fields into a concise output message, not to mention error handling, is dealt with by the common code in the parent class.

C. Computer-initiated requests

The SMS server's facilites can be accessed by other computers on the network via a simple socket-based API. The client connects to the server on a well-known port and transmits the recipient's phone number and the message to be sent. This can be done from any programming language that can talk to sockets.

To make it more convenient to call the SMS server from shell scripts and similar glue languages in which it is easy to spawn external processes but comparatively hard to interact with sockets, a basic command-line utility called `sms-send` (which itself talks to the socket API) provides equivalent functionality. It accepts the text of the message either on its standard input (handy for building pipelines of commands) or as a command line argument.

One of the clients of this API is a CGI script sitting behind a form that can be used to send a short message from a web browser. Another one is the gateway from mail to SMS: email sent to a special address will be transformed into a short message and sent to the phone number specified in the subject[2].

1. "Push" services: the watchers

The most interesting family of clients of this API, though, is that of "push" services. These are little programs that send short messages to the user's phone when something interesting happens, without the user having to manually poll for the information.

1. The *share watcher* is triggered at regular intervals (e.g. three times a day) by the `cron` system scheduler; it then fetches the stock quote of interest from the web and checks if it is in the "alert" range(s) specified by the user. If so, a short message with the current quote is sent to the user's phone. Otherwise, the quote is ignored and nothing is sent.

2. The *news watcher*, too, is triggered by `cron`; but, instead of checking stock quotes on the web, it checks news items on teletext. The latest news item at the time of invocation is scanned for specific keywords that the user is interested in. In case of match, the item is forwarded to the user's cellphone. If no match is found (or if the same news item has already been sent), the item is ignored and nothing is sent.

3. The *mail watcher* is triggered by email being delivered to the user's account. The user sets up a series of filters that, based on the value of particular header fields (e.g. the sender is the boss), determine which messages are to be forwarded to the cellphone.

[2] Due to the size constraints of SMS, only the sender, subject and first few characters of the body are actually sent.

As the reader will imagine, this list can be easily extended by any user of the system who cares enough about a personal pet feature to spend an afternoon to code it up using the available building blocks, as indeed happened with the "access log watcher" that sends you a short message when your program has been downloaded another thousand more times.

It is also worth noting that most if not all of the handlers described in the "pull" section can readily be transformed into watchers if desired by simply running them from a `cron` job and wrapping them into some logic that only sends out their output if it meets certain conditions.

III. Security issues

We believe that a reliable system should have security designed in from the start, rather than as a last-minute add-on; however we also believe that, when the system is still at the prototype stage, putting too much emphasis on the security aspects may get in the way of the brainstorming and experimentation, preventing the creators from trying out new ideas because of the effort required to implement them with ironclad security.

We thus did not regard security as a primary goal but we take this opportunity to observe the potential threats and desirable security features for a system of this nature.

A. Global safeguard and user identification

The first and most likely threat that we wanted to guard against did not come from attackers but from bugs. Since the phone company charges us for every short message we send, we wanted to prevent errors in the plug-in components such as watchers and handlers to be able to cause the server to send out thousands of unwanted messages. To this end, we added a low-level safeguard in the core of the server's code: the "send message" method of the phone object refuses to work unless the "allowed outgoing messages for today" counter is above zero; this global counter, which applies to the activity of the entire server, is decremented every time the method is invoked and is only reloaded at midnight or when the server is launched.

Some basic form of access control was also desirable. For the first implementation we considered it sufficient to base the identification on the phone number of the caller as supplied by the cellular network in the header of the short message, and we did not worry about attackers with the ability of forging SMS headers. The server holds a configuration file with a list of known phone numbers, each assigned to a class of phones such as "staff" or "friends". Each class is then assigned a set of privileges.

B. The quota system

A system of quotas is in place to limit the resources that each user may consume. For each class the SMS server administrator can set the maximum number of messages that each individual user in the class can send in a day, as well as the

maximum number that all the members of the class taken together can send (and similarly for receiving). The "friends" class (for people outside our company that we allow to use the system) has a reasonable quota of 10 messages/user/day and, in a year of use, we have not seen anyone exceed this limit[3]. The "unknown" class, which covers any phone not explicitly listed by number, has low but non-zero quotas so that we can demonstrate the system to anyone by using their own phone. If they are interested, we can upgrade them to "friends"; in the hypothetical case of abuse, we can always downgrade them to "barred", which has quotas of 0. Given the exploratory nature of our project we find that assuming people to be trustworthy by default serves us better than adopting a strict security policy where nobody can use the server unless explicitly authorised.

The "staff" class has rather high quotas (20 messages/user/day): we want members of our staff to be free to use the service as much as they like, even from programs, while still being protected from bugs in their own experimental software that might cause an infinite loop sending out messages one after another. In such a case the quota system protects the user by eventually blocking service to their phone while the server continues to work normally for everyone else (unlike what happens when the global low-level safeguard is triggered). Since we intended that, for staff members, the quota ought to be a protection but not a limitation, users in this privileged class have access to a special command that allows them to manually reload their quota, from the phone, when it runs out. Note that the quota system operates at a higher level than the basic safeguard mentioned previously: that one cannot be overridden by anyone other than by killing the server process[4].

The alert reader will have noticed the anomaly of a staff phone that runs out of quota and sends in another message to reload its quota: why isn't the "reload" message itself rejected, since the quota has run out? A similar paradox becomes apparent when the class quota runs out and a user, whose personal quota is still intact, has its request rejected: the server should definitely tell the user why this request can't be serviced (the user can't possibly be expected to know that the *class* quota has been exceeded), but this explanatory reply can't go through if there is no quota left. We solve these problems by introducing two types of message, normal and special. All messages are normal except "you've run out of quota" (from server to client) and "reload" (from client to server), which are special. The quotas mentioned above apply to normal messages, but special messages have separate quotas. In most classes there is a quota of one special message from server to client, through which the user whose normal quota runs out can be told (but not more than once) that this has happened. The quota of special messages from client to server is 0 for all classes except "staff", which prevents any other class from performing the "reload" operation.

C. Denial of service attacks

The structure of the quota system has been designed to guard against various denial of service attacks. The presence of the global safeguard counter prevents malicious users from making us pay an unlimited bill, but could allow them to bring the server to a halt if they managed to send enough junk messages that the server's responses exhausted the global counter. However, no single user can do this because each user has an individual quota that can't be exceeded. Even if a great number of malicious users were to all send junk messages to the server, each within her quota, they still wouldn't be able to bring it down because they would first exhaust their[5] class quota, leaving the server available for the other classes of users.

There could still be a denial of service attack staged by a large group of unknown users: first, a subgroup of them sends out enough junk messages that the class quota for "unknown" is exhausted; then, one by one, new unknown users send one junk message each, forcing the server to send out a "sorry, quota exhausted" reply (which, being "special", isn't subject to the class quota limit as long as that particular recipient hasn't received it before). The trick being exploited in the attack is that the "unknown" class, unlike the others whose members are explicitly listed, does not have an upper limit on the number of members; consequently, assuming that there is no limit on the number of phones that may be used in the attack, the server can be tricked into sending out "sorry, quota exhausted" messages until it hits its own global safeguard and shuts down.

After noting in passing that the attackers would have to pay a combined phone bill as high as that that they would impose on the server, we observe that this attack can be made inoperable by setting the quota of special messages from server to client to 0 for class "unknown". This in other words means that the server will only send the polite explanatory "sorry, quota exceeded" message to those phones that are explicitly listed in its configuration file; unregistered phones will instead be silently ignored when their quota is exceeded—and this seems fair enough.

D. Logging

A simple but effective weapon in our security arsenal is a logging facility that records all the invocations of the low level operations of the server together with their parameters, including header and body of all messages sent and received. This is effective both as an intrusion detection tool and as a debugging facility. While it is true that this may violate the users' expectation of confidentiality about their email or share portfolio, it is in fact rather common to grant the administrator access to the Trusted Computing Base; after all, users ought to be accustomed to their sysadmin being able to read their email and private files. One user, our colleague Ken Wood, suggested to trivially encrypt (e.g. with ROT-13) the body of the short messages before dumping them to the log—not to add any confidentiality but to make the ethical point that, while the administrator is under a duty to regularly monitor the log for anomalies and may need to be able to read any of the messages in the log to detect a particular problem, it should be made clear that the administrator has no right to casually read the bodies of the messages unless there is a specific need for it.

[3] Part of this may be due to the fact that users, too, have to pay their own phone bill for the short messages they send to the server.

[4] Or, more deviously, by changing the system date.

[5] Hopefully these conspiring bad people would all be from the "unknown" class. But, wherever their provenance, we'd clearly move them to the "barred" class as soon as we found out!

IV. Personalisation

A. Personal handlers

While the push services are clearly personalised, since each user individually sets them up in `cron` with the desired preferences, the pull services are, in the system so far described, the same for all users. With a little but important change we made the server substantially more flexible by introducing personal handler directories. Again similarly to what happens in a web server, users can now have an `sms-bin` directory in their home directory and when a handler is invoked it is searched in there before checking among the system handlers. This allows the user to add new commands and also to substitute (hide) system commands if desired.

The reference to the "home directory", and the assumption that users of the server have a home directory on our system, is a unixism that we adopted merely for its convenience. It relies on a mapping between phone numbers and userids, which we provide in the same configuration file that lists the class of each registered phone. Conceptually, though, users (think of people in the "friends" class) should not be required to have a full login account on our system in order to be able to set up personalised handlers: the only necessary items would be a storage place for new binaries and a mapping from the phone number of the caller to a list of personal handlers to be searched before the system ones; there is no *inherent* dependency on the underlying OS's structure of user accounts. In practice, however, one will want to allow these users to edit the list of handlers and to add and remove binaries from the storage place, all subject to some sensible form of access control (and indeed *access*!). All in all, relying on the login account structure of the underlying OS seemed to be the most practical solution, as well as the one that users would find most natural.

B. Extreme mobility

Going back to the "caravan or hotel" alternative, an even more extreme idea than using the cellphone that the user is already carrying is... letting the user use *someone else's* cellphone, without having to carry anything at all. So we designed an alternative authentication architecture to support this idea.

The brute-force solution would be to ensure that each message from client to server contain, as well as the command itself, a userid and password pair through which the server can establish the identity of the caller regardless of the particular handset being used. If one were to worry about replay attacks from GSM eavesdroppers (or, perhaps more probably, from the owner of the borrowed handset in the memory of which the user might have inadvertently left the short message it just sent), a variant of this solution based on a one-time password scheme [7] might be adopted.

Anyone who ever used a cellphone's keypad to enter an SMS message will however testify that having to enter a userid and a (suitably long) password *for every command* cannot be rated as a very practical solution. We thus designed a system that, while user-centric instead of phone-centric to support mobility between handsets, would not require authentication at every transaction and would be friendly towards the relatively common case of anonymous users.

With this new scheme, privileges and resources are allocated to users, not to phones: it is now users, not phones, that belong to the classes described in section III. However, when a user performs the authentication procedure[6] by sending a short message of the form `login` *`userid password [hours]`*, the server remembers the phone from which the user was calling and assumes that, for the duration of a configurable "activity window", any messages from that phone genuinely come from the above mentioned authenticated user.

The time duration of the activity window, which has a sensible default (10 hours) and a maximum value (2 days), can be set by the user while authenticating. It can also be cut short by the user "logging out" of the phone with a special command (to be issued e.g. when a borrowed phone is returned to its owner) or by another user authenticating from the same phone. When outside the activity window for a given phone, the caller is assumed to be the "anonymous user" for that phone (each phone has its own). Thus, for every incoming message, the server always has a clear notion of who the caller is: either an authenticated user or the anonymous user for that phone.

Every user, including the anonymous users, is assigned to a class; for known users and for anonymous users of known phones this mapping is performed explicitly by the server administrator; for anonymous users of unknown phones, instead, with a natural extension of the previous strategy, the assignment is to a default low-privileged class called "unknown". Classes define quotas and lists of commands accessible to users in the class.

Each user also has an additional list of "personal" commands; these can be new executables or aliases for existing handlers, possibly with additional default parameters (for example I may alias a short command name to the existing share handler called with the symbol for my favourite stock and with the price at which I bought it three months ago, so that the handler can return not just the current stock price but also the percentage profit or loss). For simplicity and uniformity there is no separate alias definition language: aliases are implemented as one-line shell scripts that call the relevant handlers, so in fact *all* the personal commands can be seen as new executables.

The benefits of this scheme are as follows: users can freely access the server through any available cellphone, even if they don't own one; they don't have to continually retype their passwords during normal usage; and those who own a phone may, if they so desire, delegate most of their privileges to the anonymous user of their phone, reserving password-based authentication only for the truly sensitive services and for those depending on extensive personalisation.

C. Customising the handset

The Smart Messaging [8] enhancement proposed by GSM manufacturer Nokia provides, among other things, a way to redefine the menus of your Smart-Messaging-compliant cellphone by sending special messages to it from a previously designated phone.

We obtained a few such handsets and nominated our SMS

[6]Here too, the use of a one-time password scheme is possible and selectable on a per-user basis depending on the individual user's stance in the trade-off between convenience and security.

server as the accepted source for updates. We then wrote a utility allowing a user to reprogram the menus on her handset from her workstation. With this arrangement, the personal commands referred to in the previous subsection (i.e. invocations of handlers with specific options preloaded, such as `RAIL CAMBRIDGE LONDON` to enquire about the time of the next train to London) can now be downloaded to the phone. The user can now recall them by browsing through a list of choices instead of having to type out the command and its options in full.

This level of handset personalisation could in theory be coupled with the user authentication architecture previously described, loading a new set of menus in the phone whenever an activation window is opened (login) or closed (logout). But storing (or worse *overwriting*) state, and especially menus, in a borrowed phone would rightly be felt as an intrusion by the legitimate owner. (As a matter of fact, it would also be impossible without manually enabling the handset to accept smart messages from our SMS server.) We thus believe that, unless the smart messaging specification is extended to explicitly allow multiple users per handset with each user's state held in a protected area[7], it would be inappropriate to attempt to teleport a user's own interface to someone else's phone. We thus currently only support customisation for the handset's owner.

Another way in which Smart Messaging can enhance our server is by providing extended responses when the destination phone is known to accept Smart Messages. The Active Badge handler, which returns the location and nearest phone of a colleague, could return the phone not just as a string but as an active button that one could press to initiate the voice call.

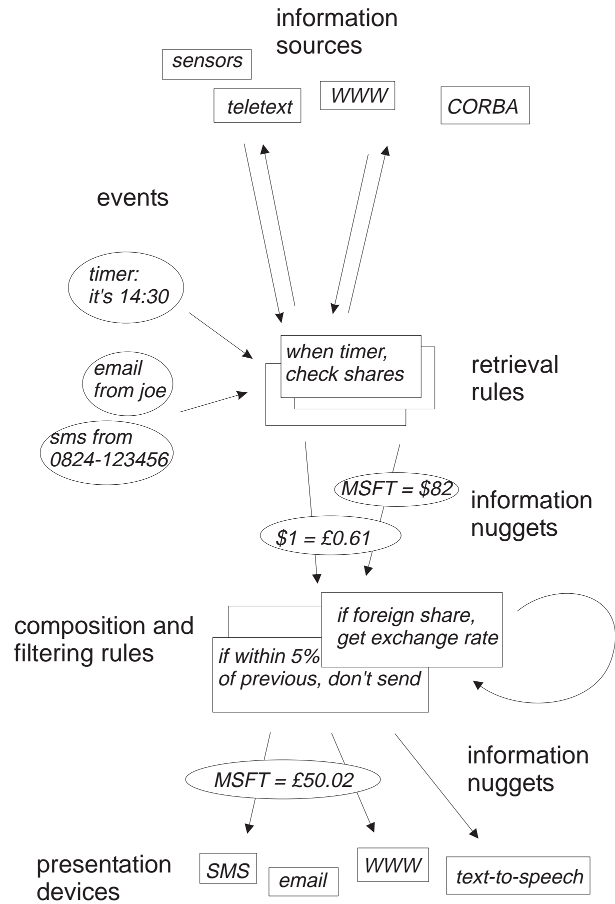
Endowing the handset with some intelligence is a field that attracts considerable commercial interest, and Nokia's Smart Messaging is not the only offer. Unwired Planet [9] proposes a similar architecture in which the handset is even more customisable, to the point that it becomes a miniature web browser. We haven't tried this yet, simply because the technology was not yet as easily available as Nokia's, but our system is open and will readily interface to whatever devices will prevail in the marketplace.

D. The Personal Information Service

The experience gained with the SMS server opens up new horizons and leads us to consider the system we built as an exploratory prototype of a wider-ranging system which we call the Personal Information Service.

In the new system that we propose, a multitude of *information sources* (world wide web, teletext, CORBA services, sensor systems, etc) may be consulted according to *user-defined rules* in response to *events* (timer, email arrival, incoming SMS, sensor output above threshold, etc) so as to produce *information nuggets* that, after various filtering and composition stages, will be delivered to the user through one of many *presentation devices* (SMS, of course, but also email, personalised and self-updating web page, nearest computer display, pager watch, telephone via text-to-speech synthesis, actuators of various sorts, etc).

[7] Not that we are advocating this: it would probably add too much complexity.



The following points are noteworthy:

1. Pull and push must both be supported. In the above outline, both the user requests and the data-driven activations are ultimately seen as "events" that trigger the extraction of information nuggets from information sources.

2. The effectiveness of the system is greatly enhanced when data from various independent sources is combined to provide new information, like we did when joining the stock prices from one web site with the currency exchange rates from another to give the user the net profit or loss in local currency for foreign shares.

   To enable these manipulations, the data extracted from the various information sources needs to be "transduced" into information nuggets represented in a common format. Based on our experience writing handlers and generic library classes to support the creation of new handlers, we recommend that this be some form of associative array mapping field names to values (preferably typed). This could in theory be wrapped up as an object, but we would tend to warn against it: encapsulation is likely to make it harder, not simpler, to merge two nuggets into one, and we see the intelligence best placed in the rules that know how to combine several nuggets rather than in the methods of each single isolated nugget.

3. The personalisation element is essential and will have to appear at several stages. Firstly, the user must be able to designate and schedule sequences of events that may generate information; this is what is done in the SMS server

when the watchers are set up: a cron job schedules invocations of the share watcher and a mail filter intercepts incoming email. Secondly, the user must be able to define rules specifying what to do when events occur and generate nuggets ("whenever a share nugget arrives, if it's for a foreign share, fetch an appropriate currency nugget and combine them in this way") as well as filters deciding which nuggets are interesting and which ones should be thrown away. This is going to be a non-trivial piece of work and will require choosing a suitable language in which to express the rules. We would advise against the temptation of creating an ad-hoc language from scratch; we recommend instead the adoption of an existing high level embeddable scripting language [10] such as Tcl [11] or Python [6], appropriately augmented with a few primitives to handle the rules. But, even so, we feel that any rule manipulation language that is sufficiently expressive to be useful will be too complex for non-technical users to manipulate directly. We envisage a system in which "rule programmers" write libraries of generic rules from which non-technical users can select, parameterise and combine the appropriate ones as desired, perhaps through a visual interface[8]. Thirdly, the user must be able to specify rules to determine the preferred presentation device based on the newly created nugget and on environmental information such as the time of day, the last known position and connectivity status of the user and so on.

V. Conclusions

The SMS server demonstrates how even a system with very low input and output bandwidth such as SMS can be used to provide the mobile user with valuable, non-trivial services. Personalisation provides extra context thanks to which even a very short message can yield a high information content.

We have built a personal information gathering system with feeds from a variety of sources. The ubiquity of the cellphone makes this system accessible from anywhere.

We see our project as anticipating many important features of more sophisticated systems to come which will filter and combine information from many sources according to personalised rules and deliver it through a variety of communication devices. We have been working towards the most globally accessible service that we could implement, aiming at what we call *Global Smart Personalisation*.

VI. Acknowledgements

We are grateful to our many colleagues who have been using our system regularly since its deployment, providing valuable stress-testing and user feedback. In particular we thank Steve Hodges for inventing and implementing some handlers and watchers early on and for eventually taking over the administration of the server, Frazer Bennett for assistance in interfacing to the Active Badge system and Quentin Stafford-Fraser for adding support for Nokia Smart Messaging and personal handler directories.

VII. About the authors

Frank Stajano (`http://www.orl.co.uk/~fms/`) obtained his Dr. Ing. degree (electronic engineering) from Rome in 1991. He has been a research scientist at ORL since 1992, working on distributed multimedia, GUIs, reusable components, databases, internet applications and other topics, always with a strong interest in scripting and OO. He is currently with the Computer Security Group at Cambridge's Computer Laboratory, investigating security aspects of ORL's Piconet project.

Alan Jones is a principal research engineer at ORL. He was awarded a B.A. in physics in 1981, a computer science diploma in 1982, and a Ph.D. in computer science in 1986, all from Cambridge University, England. He has worked on control systems, sensor devices, optimization, simulation, real-time multimedia, and a variety of mathematically oriented projects. Current interests include communication and navigation systems, as both a keen user and a developer.

References

[1] Tristan Richardson, "Teleporting in an X Window System environment", *IEEE Pers. Comm.* No. 3, 1994.

[2] Kenneth R. Wood, Tristan Richardson, Frazer Bennett, Andy Harter, Andy Hopper, "Global Teleporting with Java: Towards Ubiquitous Personalised Computing", *Computer*, Vol. 30, No. 2, 2/1997.

[3] Tristan Richardson, Quentin Stafford-Fraser, Kenneth R. Wood, Andy Hopper, "Virtual Network Computing", *IEEE Internet Computing*, Vol. , No. 1, 1-2/1998.

[4] Roy Want, Andy Hopper, Veronica Falcao, Jonathon Gibbons, "The Active Badge Location System", *ACM Trans. Inf. Sys.*, Vol. 10, No. 1, 1/1992.

[5] ETSI, "Digital cellular telecommunications system (Phase 2+), Technical realization of the Short Message Service (SMS), Point-to-Point (PP)", GSM 03.40 version 5.4.0, 11/1996.

[6] Guido van Rossum, *Python Library Reference*, *Python Reference Manual*, May 1995, CWI Reports CS-R9524, CS-R9525.

[7] Leslie Lamport, "Password Authentication with Insecure Communication", *Comm. ACM*, Vol. 24, No. 11, 11/1981.

[8] Nokia, *Smart Messaging Specification*, 1.0.0, 9/1997. http://www.forum.nokia.com/

[9] Unwired Planet, *Getting Started with the UP.SDK*, V. 2.0, 1998. http://www.uplanet.com/

[10] John K. Ousterhout, "Scripting: Higher Level Programming for the 21st Century", *IEEE Comp.*, 3/1998.

[11] John K. Ousterhout, *Tcl and the Tk Toolkit*, Addison-Wesley, 1994.

[8] Note that this does not preclude technically-minded users from writing their own rules if they so wish; in fact, from our experience, we recommend that the rule manipulation language and API be accessible and well documented.