Give Me Back My Personal Mainframe!
Lessons Learned from Plan B and Position Statement

Francisco J. Ballesteros — Enrique Soriano Salvador
11/13/2006
Laboratorio de Sistemas — Universidad Rey Juan Carlos
{nemo, *esoriano*}*@lsub.org*
Madrid, Spain.

ABSTRACT

Trends in system research move more and more towards a peer to peer world. Complex issues in distributed systems are eagerly addressed, the more complex the problem the better. We work to develop a highly distributed, dynamic, and heterogeneous computing environment. We have been using Plan B, a modified Plan 9 system that adapts to changes in the environment, and learned some lessons by using it. Some ideas worked nicely, other ones did not. In the end, we ask ourselves: are we on the right track? Or are we going to learn from Google?

1. Introduction

Today computing environments are complex. They are made of lots of devices and machines interconnected through multiple networking technologies. They are heterogeneous, dynamic, must support network partitions, have administration problems, run different sets of system software, have different applications and capabilities, and the list goes on.

Research on system issues is eager to address different aspects of most, if not all, of the problems mentioned. There is work on portable environments to develop applications that could work on different platforms [4, 10, 14]. There is work on technologies to organize the miriad of devices without having to rely on central servers or machines [5, 7, 16]. Others addressed how to move services and applications [9, 12]. Yet more work is targeted at making different applications interoperate [8, 15].

Along these lines, our own past work involved building a system that could work well for heterogeneous, distributed, and dynamic environments. We modified Plan 9 and constructed what we call Plan B [1]. The resulting system was simple and adapted well to changes in the environment.

But after thinking it twice, and looking back at what worked well in Plan B and what did not, we argue that many problems addressed by may research projects are not problems indeed. Furthermore, we argue that such problems arise from considering a *distributed* system made of *heterogeneous* and *volatile* components. The resulting system must be able to cope with the problems resulting from these properties.

But we now think that on a globally connected world, the environment is no longer distributed, is no longer heterogeneous, and is no longer volatile. Soon, we will have ubiquitous access to the Internet. In this scenario, a complex peer-to-peer architecture to enable access to our own resources makes no sense. Peer-to-peer operations are necessary in partitionable systems, mainly to allow the user to keep using the system at isolated network partitions. But our experience with Plan B shows that this scenario is

not frequent even at present time. Moreover, we think that this problem will vanish soon. We suppose that the system is not partitionable anymore. This is the key to understand the approach we propose.

Google [6] provides an excellent example that suggests that we do not have to pursue distributed, heterogeneous, and dynamic environments. It centralizes the implementation (at least when seen from the outside), and provides ubiquitous access to it. Users open their browsers, point them to *pages.google.com* or to *calendar.google.com* and start working. Their computers are only terminals accessing to a centralized service. The service is centralized at google, and the user does not notice it. That is because nowadays we assume continuous connection to the Internet at home. Soon, we will have continuous connection to the Internet at any location. In a sense, Virtual Network Computing was a pioneer in this respect, and pursued a similar model at a much smaller scale.

The google file system [6] is another example. It centralizes control, to handle in a much more simple and centralized implementation the miriad of distributed storage and computing devices used to implement its service.

The same underlying idea could be applied to the system software. We could stick to a single system, yet permit the system to span all the distributed devices of interest. Our position statement is that to build the kind of environment we envision, **we only have to combine all the I/O devices we need and plumb them to a central, per-user system.** Like we did when using mainframes and connected terminals. The only difference is that computing is so cheap that now we may have one mainframe (read: PC) for each user.

2. Plan B

Plan B builds heavily on Plan 9. Indeed, a 4th edition Plan B system is built by executing user-level programs on a standard Plan 9 installation. Besides the design principles inherited from Plan 9, Plan B added these ones:

1 File interfaces used to export resources correspond to resource interfaces with a high level of abstraction.

2 Users do not specify (usually) which resources they want to use (mount), just what properties must hold for such resources.

3 Resources corresponding to the physical environment (e.g., power switches, location, and user context information) are considered as standard devices in the system.

The first point is easy to see by comparing the interface provided by the Plan 9 window system, Rio, and the Plan B window system, Omero. The former exports images and windows. The latter exports widgets. The file hierarchy provided by Rio provides a powerful interface, but does not expose the structure of the interfaces shown. On the contrary, the file hierarchy provided by Omero mimics the hierarchy of widgets on the screen, providing more control to external programs interested in handling the different GUI elements. This is further described in [2].

The second point is important when resources are dynamic or mobile. Plan B name spaces mount devices that meet a certain criteria (e.g., their location and ownership match those of the machine). The `bns` program provides an indirection between the actual file systems (devices) used and the file tree seen by the user's name space. See figure 1. When a device ceases operation, another one is selected instead. When no one is available, an empty directory takes the place of the device's file tree.

As an aside, we have to say that open file descriptors for files reached through `bns` are kept untouched. Thus, applications do not get confused when a device is replaced with another. However, when a device becomes unavailable, `bns` reports I/O

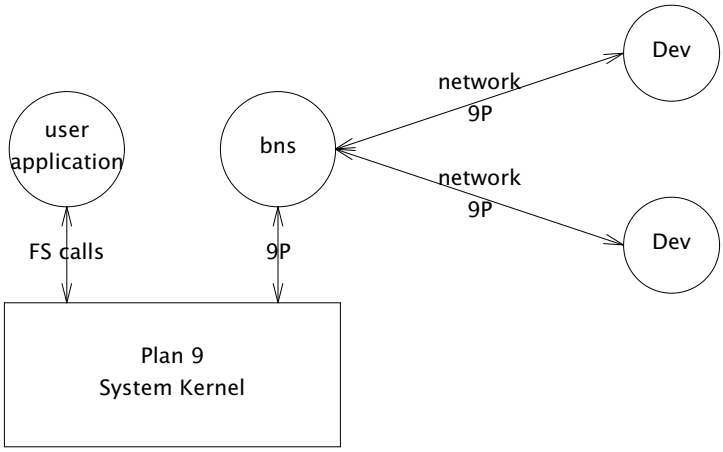

Figure 1: Bns *provides resources by relying on external device file trees.*

errors immediately, instead of waiting for the utterly long TCP timeout.

The third point is important when using the system for environment automation and to build smart spaces [15], but is otherwise uninteresting. As an example of how the three ideas work together, the next few Rc lines mount the window system from any wall-mounted screen in the room where we are, mounts the light power switches for the same room, and starts a presentation:

```
# mount any UI device at 136 mounted on the wall.
# /srv/vol reaches bns and the mount spec is what we want.
; mount /srv/vol /n/ui '/devs/ui location=136 size=wall'

# mount all X10 devices known
; mount /srv/vol /n/x10 '-U /devs/x10'

# switch off the lights
; for (light in /n/x10/pwr:*136*light*)
;;  echo off >$light

# create a single top-level widget for viewing large images
; mkdir /n/ui/page:slide

# show all the slides, as the user hits return
; for (slide in $slides){
;;  cp $slide /n/ui/page:slide/data
;;  read
;; }
```

2.1. What did work

After using Plan B for several years (including previous editions of the system), we have found that there are several things that worked well:

- **Using abstract interfaces for resources**. Interfaces like those provided by Omero (with roughly a file per widget), make it easy to build programs that operate on the resource provided. We have scripts to press buttons shown in user interfaces, to recreate interfaces, etc. All of them rely on the structure exposed by the file hierarchy used to map the abstract interface for the resource. None of them required changes in the applications or the window system.
- **Using real files for application data**. Our mail reader (and composer) stores

mails as they are shown by most readers. Attachments are unpacked and kept at real files. This permits using the regular editor and a few scripts to read and compose mail. The same approach is used to keep context information (e.g., user status and device locations), we keep that information in real files, already decoded. This simplified the construction of tools without having to modify a file-server program that decodes a more compact representation for the information stored.

- **Adding location information to resources**. Along with each resource, we added a few attributes and their values. Two very useful ones are the name of the resource owner and the known location for the resource. The example Rc session above illustrates how we use this information. It has proven to be invaluable.
- **Placing an indirection between I/O devices and name spaces**. `Bns` provided a service similar to `recover` (a program from Russ Cox to recover from lost connections to file servers). However, `recover` would not be enough to let us cope with resources like audio that may be attached to different machines at different points in time. `Bns` proved to be very helpful to automatically switch from one device to another as machines were brought up and down in the room. Note that here we refer to I/O devices used for human interaction, i.e., not to disks.
- **Keeping a central registry for devices along with their attributes**. In a previous edition of the system we used a peer-to-peer discovery protocol to learn of new devices. Keeping a central registry at our main file server machine has eliminated discovery timeouts and helped us a lot to keep the list of existing devices up to date.

2.2. What did not work

But not everything went well. Here we state what did not work well for us.

- **Peer-to-peer discovery**. It simply failed. All the attempts to decentralize resource discovery failed. We think that the reason is that such protocols are very sensible to timeouts chosen to declare devices as failed. Small timeouts increase the system load and network traffic. Big timeouts decrease the system responsiveness. Besides, not all machines agreed on the set of known devices. But this was not new, just that we had to re-learn it the hard way.
- **Recovering from file server crashes**. `Bns` knows how to switch file systems, including /, and not just devices. However, to do so, it needs to resolve addresses. This required a caching `cs` service, because `bns` had to re-dial the file server when it came back without relying on ndb files. However, it was common in practice that addresses never seen before had to be resolved, and there were subtle relations between factotum, `cs`, the caching `cs`, and `bns`. In the end, we found ourselves rebooting to re-establish the normal operation of the terminals after a file server reboot.
- **Switching the root file system**. `Bns` could switch between local file systems (on laptops) and the central file server. We modified `ipconfig` and a few other programs to adapt to changes in IP addressing when moving the laptops, in the hope that `bns` could automate switching between the central file server and the local disk. However, user paranoia arises. We found ourselves asking the system: where's / coming from? And that happens many times. Automatic switching for non-essential file systems worked well, including devices and the file system with MP3 files. It jut did not work for the root file system and for the user's home directories.
- **Having to use a Plan terminal to use the system**. In many cases, opening a browser in a suspended system and using Gmail was faster (hence more convenient) than booting a Plan B terminal just to read the mail. Using `drawterm` to a CPU server might have been a choice, but we do not run `bns` on CPU servers as of

today.

3. Which ones are the problems?

Let's reconsider several important issues usually identified as problems, regarding the system any of us would like to use.

Distributed management of resources is a though issue. It leads to complex topics like distributed garbage collection, distributed programming, dealing with failures in part of the applications and the system, etc.

If we maintain centralized applications that can be used from anywhere else, like Google does, distribution becomes a non-issue. To say it in a different way, we have to use distributed devices, but we do not have to distribute the system or the application.

Heterogeneity of software and hardware platforms seems to be a problem because we must develop applications (or provide platforms to let others develop them) that run on a wide variety of systems and architectures. Again, centralizing the entire system (but for devices) may greatly simplify this.

Heterogeneity of devices has been dealt with for a long time. We have (almost) abstract interfaces that hide most of the complexity of the underlying devices. Most (if not all) systems do this. Therefore, it is reasonable to think that we might do the same when the devices are scattered across the Internet. This might be a serious problem for efficiency. However, using high-level interfaces (e.g., like we do for graphical user interfaces) greatly reduces the needs in latency and throughput for using a remote device.

Heterogeneity of machines becomes void when we use a single system. We could rely on machines that may be running a different system, and use them as if they were hardware, and borrow a few devices to reach our central system. Inferno [4] might be a powerful tool for achieving this.

Dynamism of resources and services is a huge problem because it forces our distributed systems to adapt. If they do not adapt, they may become useless or broken. However, we are starting to know how to handle devices that may be plugged or unplugged while the system runs.

4. What do we want?

What we want is to **use all the variety of devices that we have with our central system.** What did we do with our personal computer? We simply plugged devices into it. Then we used the system to carry out the task at hand. Why can't we do the same now?

Computing is so cheap that we can afford losing all or most of the computing power of most of the computers we use. We only want the computer to carry out our commands fast. But that happens today if we use even a single computer. If we waste most of the power of all but one (or a few) of the computers we use, nobody would notice.

However, we still want to use all the devices we need. And to borrow them for a while. We want our system to be able to operate with devices found at any other machine. Although we would not notice that the processor of the machine we are typing at is wasted, we would notice if we cannot use its DVD reader to copy a movie into our system and reproduce it at a nearby screen.

5. The Personal Mainframe

We advocate for a personal mainframe. More precisely, a personal CPU server for each user. The new personal computer would be just one (or a few machines at most) that implement a computing system for the user in a centralized manner. We already know how to build these kind of systems. To make this system useful today, we must make it available at any place and we must let it employ whatever devices the user might

consider of interest, be they local or on the antipodes. There are a few differences between this, new, personal computer and traditional ones:

1 Instead of a system bus we have now the Internet to interconnect all the devices of interest to our system.

2 The computer is personal, it is not shared, because there are many computers per user. (There are three at my house and six at my office right now). Of course, a central file server and a few other shared services may provide common services to several personal servers.

3 Devices *are* shared. DVD readers (and players!), screens, keyboards, phones, disks, and all other devices can be potentially shared. We may want to hand a keyboard to a different user (e.g., in the living room to control a Set-Top-Box), or to insert our DVD into a reader nearby. At different points in time (may be simultaneously) a device might be used by different users.

6. Building the Personal Mainframe

To build this computer, we need a bus to export all the devices to our system. In Plan 9 [13] and Plan B [1] we have such bus. The 9P file system protocol provides a simple way to export resources like devices across the network. Plan B has used this bus to export devices using abstract interfaces, so that the network latency and/or bandwidth problems get minimized. For example, Plan B uses MP3 as the native audio format, instead of PCM samples.

Other resources, like video, require more work. Perhaps approaches like [17] that cut data paths from sources to sinks would be of help. We want to keep control centralized in the system (as well as the implementation of the application). However, this does not mean that data must go from a device nearby to the system just to go back to another device close to the former.

We need a way to adapt to changes is the set of devices available. Much work in adaptable systems (e.g., Odyssey [11], Plan B [1] and Gaia [3]) can help in this respect. In Plan B devices are plugged/unplugged from the application name space depending on the user preferences and the availability and properties of the devices. But more work is needed. At the very least, implementations for Plan B devices in Inferno are required, to export devices from underlying systems like Windows and Linux.

We need to explore how to build adaptors to make this approach system-neutral. For example, Windows users would want other DVD readers to exhibit the same interface provided by the native device. At least, they want an interface as close as feasible. Therefore, work is needed to build adaptors for systems that we want to use, to adapt devices from others so they could use them.

References

1. F. J. Ballesteros, E. S. Salvador, K. L. Algara and G. Guardiola, Plan B: An Operating System for Ubiquitous Computing Environments, *Proceedings of IEEE PerCom. Also at http://lsub.org.*, 2006.
2. F. J. Ballesteros, G. Guardiola, K. L. Algara and E. S. Salvador, Omero: Ubiquitous User Interfaces in the Plan B Operating System, *Proceedings of IEEE PerCom. Also at http://lsub.org.*, 2006.
3. E. Chan, J. Bresler, J. Al-Muhtadi and R. Campbell, Gaia Microserver: An Extendable Mobile Middleware Platform, *Proceedings of 3rd IEEE Intl. Conf. on Pervasive Computing and Communications*, 2005, 309-313.
4. S. Dorward, R. Pike, D. L. Presotto, D. M. Ritchie, H. Trickey and P. Winterbottom, The Inferno Operating System, *Bell Labs Technical Journal 2*, 1 (1997), .
5. W. K. Edwards, M. W. Newman, J. Sedivy, T. Smith and S. Izadi, Challenge:

Recombinant Computing and the Speakeasy Approach, *8th ACM Mobicom*, 2002.

6. S. Ghemawat, H. Gobioff and S. Leung, The Google File System, *19th ACM Symposium on Operating Systems Principles*, 2003.
7. S. D. Gribble, et al. The Ninja architecture for robust Internet-scale systems and services, *Computer Networks. Special issue on Pervasive Computing 35*, 4 (2000), .
8. K. Luyten, *An XML-based runtime user interface description language for mobile computing devices*, LNCS, Springer, 2001.
9. D. Maltz and P. Bhagwat, MSOCKS: An Architecture for Transport Layer Mobility, *INFOCOM*, 1998.
10. S. Microsystems, Java 2 Platform Standard Edition, API Specification.
11. B. Noble, M. Satyanarayanan, D. Narayanan, T. J.E., J. Flinn and K. Walker., Agile Application-Aware Adaptation for Mobility, *Proceedings of the 16th ACM Symp. on Operating System Prin.*, 1997.
12. T. Okoshi, M. Mochizuki, Y. Tobe and H. Tokuda, MobileSockets: Toward Continuous Operation for Java Applications, *Intl. Conference on Computer Communications and Networks*, 1999.
13. R. Pike, D. Presotto, K. Thompson and H. Trickey, Plan 9 from Bell Labs, *EUUG Newsletter 10*, 3 (Autumn 1990), 2-11.
14. A. Ranganathan, S. Chetan, J. Al-Muhtadi, R. H. Campbell and M. D. Mickunas, Olympus: A High-Level Programming Model for Pervasive Computing, *Proceedings of 3rd IEEE Intl. Conf. on Pervasive Computing and Communications*, 2005, 7-16.
15. M. Roman, C. K. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell and K. Narhstedt, GaiaOS: A middleware infrastructure to enable active spaces, *IEEE Pervasive Computing Magazine*, 2002.
16. uPnP-Forum., Understanding uPnP., *www.upnp.org*, 2004.
17. The Box: A replacement for files, *Proceedings of HotOS-VII*, 1999.