Abstract

This paper describes a Stateless computing environment, wherein items such as documents and application preferences (a user’s ‘state information’) is decoupled from the physical computer. A Stateless system provides consistent access to users’ workspaces despite location and without error-prone and time-consuming re-configuration for each specific workstation. Toward this end, the goals necessary for such a system are outlined, and current technologies are evaluated for their abilities to realize that goal. The Model/View/Controller concept is suggested as a reasonable base for new administrative aids to overcome current tools’ deficiencies. These new programs replace some existing structures, but are designed to minimize unnecessary change and coexist with current environments.

Note: throughout the text of this document, Stateless is evaluated when administering large numbers of workstations. There is no reason to believe the same concepts could not be used when dealing with large numbers of servers. With slight modification, Stateless could also be used in the cluster administration, a highly important secondary goal.

1 Goals

To comply with the overall goal of separating user state from the method of interaction, a successful Stateless environment should achieve the following goals:

1. Despite location, a user can access his data according to his own configuration.

2. The user should have full access to the power of the chosen interface, rather than a ‘least common denominator’. For example, if a user chooses a Windows workstation, he should not be limited to a website-based content management system for access to his critical data, as this ignores expected Windows features such as access to the Windows file manager or many drag-and-drop interactions.
3. Administrators can easily add or remove software on all workstations.

4. The same tools can manage computers using different operating systems.

5. Failed updates should not render the system (or any application on it) unusable.

2 Analysis of Current Tools

2.1 Terminals

As the popularity of interactive timesharing computers rose, the computing world accommodated its larger population with terminals — multiple interface devices connected to a single shared computer. Although computing power has become much cheaper, and personal workstations have often replaced the centralized behemoths of old, terminals’ venerable history in computing continues on into the present day. By way of products and projects such as Linux Terminal Server Project\(^1\), Microsoft’s Terminal Services\(^2\), and Sun’s Ray Station initiative\(^3\), both hardware and software terminals see a range of companies and independent developers still interested in thin clients for modern networks.

Terminals and thin clients’ strengths usually lie in their ease of administration. Few computing devices act as much like an appliance as terminals do; they rarely require software updates, and the software that drives them has, on the whole, matured to the point of reliability.\(^4\) In a terminal-centric computing environment, the overwhelming fraction of important software and information resides on a single computer, which is generally easier to administer than distinct installations on multiple workstations.

However, terminals come with some distinct disadvantages. A terminal maintains a constant connection with its central computer; if this connection fails or is not present, the terminal becomes useless. Users have come to expect minimal interruptions in a wired, administered environment, but with laptops becoming more prevalent\(^5\), network connectivity can no longer be simply assumed. Using a personal computer as a mere terminal to a larger, centralized machine also misuses available resources; for instance, it ignores almost all the processing capabilities of the individual’s computer. The processor driving the archetypical terminal debuted in 1974,\(^6\)\(^7\) and modern terminals devote thirty

\(^1\)http://www.ltsp.org/
\(^2\)http://www.microsoft.com/windowsserver2003/technologies/terminalservices/default.mspx
\(^3\)http://www.sun.com/sunray
\(^4\)As an example, Microsoft’s Remote Desktop client has only one listed vulnerability, patched in 2002. (http://secunia.com/product/17/) In a similar timeframe, the latest version of Microsoft’s Office software had 10 security-related flaws (http://secunia.com/product/2275/).
\(^5\)Notebooks represent 31\% of all computers worldwide, Gazeta Mercantil Invest News, São Paulo, Brazil, 15 September 2005
\(^6\)The popular DEC VT100 terminal used Intel’s 8080 general-purpose microprocessor; see http://www.cs.utk.edu/~shuford/terminal/dec.html
\(^7\)http://www.intel.com/museum/online/hist_micro/hof/index.htm
years’ progress to a task which has hardly changed. The same principle can be applied to a personal computers’ other advantages; terminals ignore benefits such as local hard drives or multimedia capabilities.

To a user, his configuration and data will seem to ‘follow’ around the network, despite being stored on one central computer. However, unusable network conditions leaves a user stranded. Even though terminal server environments lend themselves to redundant central servers, failed updates can frustrate user access. While terminals may help ease administrative pain, their wasteful design and unsuitability for mobile work mean they cannot be used as the center-most administrative strategy to meet the stated goals.

2.2 Disk Imaging

To enforce homogenous software environments on disparate workstations, administrators can choose from a variety of disk imaging tools, which ‘carbon copy’ one computer’s software environment onto another workstation.

Like terminals, imaging tools enforce homogenous environments. Disk images\(^8\) ensure a perfect copy, mitigating the risk of a package failing to install or a setup program failing in some unexpected way. If a new disk image contains an unacceptable error, the entire previous configuration is available on archive; the administrator can simply replace the newly deployed image with a fresh copy of the old, until a new version without the flaw can be introduced.

However, the uniformity imaging tools provide comes at a very high cost. Disk images provide no way to separate programs into distinct packages or even differential updates — such a concept defeats the purpose of completely identical workstations — and so the only way to change them after deployment is to overwrite with another disk image. Many imaging tools demand identical hardware\(^9\); in most cases, this regulation stems from the operating system itself, such as the Windows Hardware Abstraction Layer.\(^10\) Disk image production also takes an incredible amount of time. For example,\(^11\) deployment of a disk image on a single machine over 100 Mb Ethernet could take from 30 minutes to an hour; a medium-sized (~50) computer lab rarely finished in under three hours, and usually monopolized several student workers and their full-time staff for a complete 8 hour work day. Larger labs could require multiple days to update. In some scenarios, such as summer days in university labs or weekends in a large organization, this downtime is acceptable, and perhaps disk images are useful in combination with some other deployment mechanism. However, the need for large periods of downtime to fix any small errors in a deployment make disk images unusable as a main method of day-to-day administration.

\(^8\) bit-for-bit copies of the original computer’s hard disk contents; not to be confused ‘software disks’, such as .iso CD images or .dmg application packages

\(^9\) the Linux Systemimager tool, for example, will not allow transfers from IDE to SCSI mass storage, and vice versa; see http://www.systemimager.org/doc/html/x237.html

\(^10\) For an example, see http://service1.symantec.com/SUPPORT/ghost.nsf/docid/1999070716282425, subsection “Differences between the source and destination computers”

\(^11\) Based on two years of personal experience under professional staff at the LSU Office of Information Technology Services, formerly the Office of Computing Services, 2003-2005
Disk imaging can be used as a way to restore a computer — and thus user state — after a catastrophe, but cannot distinguish user configuration and files from system files. The imaging process typically requires full control of the computer, and so disk images are a poor choice for constantly re-imaging a separate “user state” partition, aside from other considerations such as wasted bandwidth and time re-transferring unchanged data, and needless stress on the hardware.

2.3 Installers

Simple applications, such as the PuTTY SSH client or some high-level language scripts, can be distributed as a single file; when the user executes the program, preexisting support structures in the system\textsuperscript{12} ensure that the application runs correctly. Operating system developers may be able to support some advanced applications without extra work on the user or programmer’s part, but application complexity eventually outstrips the system’s ingrained support structures, and more involved measures often become necessary.

Consider the following environments:

1. **Apple Mac OS X**: a default Mac OS X application\textsuperscript{13} requires not only the application but serialized interfaces, localization files, metadata, and special file permissions. More advanced applications also require extra libraries. However, the vast majority of Mac OS X applications exist in a self-contained bundle\textsuperscript{14}, and require no installation. Double-clicking on the bundle in the Mac file browser launches the program; to uninstall, a user can simply delete the bundle. If multiple bundled apps require the same non-system library, it will be duplicated (one copy in each bundle). At no point in this process does a user need administrative or ‘super-user’ privileges. Some applications or libraries modify different parts of the system; these require Apple’s .pkg format, make install, or some other third party scripting mechanism as well as administrative privileges. There is no standard way to remove these systemwide modifications.

2. **Microsoft Windows XP**: Windows applications typically store information in Windows’ registry, described as a “central hierarchical database”\textsuperscript{15}. Applications also typically place their necessary files in C:\Program Files, which requires administrative privileges to modify. Because of the delicate nature of both these areas of the system, Windows applications are typically installed through a third-party installer or Microsoft’s .msi files. Through the Windows Control Panel or a separate uninstaller, applications can be removed, but incomplete uninstallations commonly occur and

\textsuperscript{12}Shared libraries, #!-invoked interpreters, etc

\textsuperscript{13}Using Apple’s recommended Cocoa API

\textsuperscript{14}http://developer.apple.com/documentation/Porting/Conceptual/PortingUnix/distributing/chapter_9_section_2.html

\textsuperscript{15}http://support.microsoft.com/kb/256986
can conflict with attempts to re-install.\textsuperscript{16,17,18}

3. **Traditional Unix-Like**\textsuperscript{19}: POSIX compliance discusses features such as threading, directory hierarchy, and environment variables.\textsuperscript{20} As one moves up the software stack to applications, there are very few agreed-upon standards for what libraries appear in any given Unix system, and none to which distributors must adhere. Installations such as Macromedia Flash or Sun’s Java Runtime Environment typically use their own minimalist and idiosyncratic scripts; they differ too much to draw any general conclusions. More so than on other systems, Unix programs are sometimes hand-compiled from source code, using a Makefile with an installation routine; this process often involves special cases for each architecture involved, necessitating tools such as \texttt{autoconf}.\textsuperscript{21} Building by source offers no dependency resolution except for a pithy failure message, often leading to further hand-compiling several more levels of dependencies. Uninstallation procedures vary from a thoughtfully provided \texttt{make uninstall} command to groveling through the file system hand-removing libraries, binaries, support files, and symbolic links.

Installers can remove the complexities of software installation (imagine a Windows administrator editing registries by hand every time a new piece of software becomes available), but they all introduce their own idiosyncratic problems. For identical machines, the concept of installed packages is somewhat unnecessary. Each independent installation is repeated work, and is a chance for flaws or mistakes to creep in. Administrators often forgo disk images’ near-guarantee of consistency for installers’ potential to complicate, since installing a package only affects the part of the system where changes are made, where disk images needlessly replace unchanged parts of the systems. Packages are also poor devices for managing user data; the task of creating new packages is hard to automate, and bears a close resemblance to scripting or programming. Installers also interrupt the user, and force difficult retraining: in short, the idea is laughable. Average users cannot be expected to manage their own state information in any of the current packaging formats.

2.4 **Package Managers and Repositories**

Because of the wide variation in Unix environments and the increasing number of non-experts using Unix-like systems, the laissez-faire installation mentality

---


\textsuperscript{17}Macromedia incomplete uninstallation notes: http://www.macromedia.com/cfusion/knowledgebase/index.cfm?id=tn_18556

\textsuperscript{18}VMWare incomplete uninstallation notes: http://www.vmware.com/support/kb/enduser/std_adp.php?pfaqid=1526

\textsuperscript{19}The family of Unix-like OSes represents many systems, but here we focus mostly on the GNU/Linux family, the BSD family, Solaris, AIX, and some of Mac OS X

\textsuperscript{20}http://www.unix.org/single_unix_specification/; http://www.opengroup.org/onlinepubs/000095399/idx/xbd.html

\textsuperscript{21}http://www.gnu.org/software/autoconf/
— sometimes totally lacking in installers — is unacceptable. Without a centralized authority dictating what system capabilities to expect, most popular distributors have aligned themselves around a group of mutually incompatible package managers, or a system of binary package installers (such as those in Windows or OS X) combined with a dependency graph. When a user elects to install a package, the system first ensures that all prerequisites are present in their correct version; if not, the program explicitly shows what packages are needed. In order to be effective, the vast majority of a running system must be registered with the package manager; for example, the entire Red Hat Linux operating system, from kernel on up, can be described as an interdependent series of rpm packages. The Debian Linux operating system dpkg operates in much the same way, using a different package format.

While package managers can point out software’s prerequisite libraries, users must still obtain and install the dependencies. To extend a package manager’s capabilities, popular systems augment their basic package managers by use of a repository, or collection of packages available over the web. The Debian-based Ubuntu Linux, for example, claims a 16,000 package repository system. Repositories are typically managed by means of a secondary tool; for instance, dpkg distributions like Debian and Ubuntu often use apt for .dpkg installation. Repository systems automatically install any dependencies; if a user were to install a word processor without having the system-wide spell-checker installed, for instance, the package manager would automatically install the spell-checker dependency before adding the word processor to the system.

The combination of package managers and repositories has become so popular that Novell, vendor of SuSE Linux, recommends that users not compile custom packages in the typical Unix fashion, but rather only use YaST, the SuSE package manager and repository combination. Systems are not limited to the vendor’s repository; apt and other tools allow users to add their own repositories, and several popular third-party repositories exist. In this way, administrators may choose to deploy applications by means of a local repository. Repositories can also be used to install full systems; for example, Red Hat uses the Kickstart system, which allows an administrator to choose standard groups of packages (or specific packages) to automate repetitive installations, and others, such as the Fully Automatic Installation project are available for different systems.

Apple and Microsoft use a less powerful technology for their automatic updating systems. Both of these systems track dependencies to a small extent; prerequisites must be installed before new software, but the tool must often be

---

22http://rpm.org
24http://www.ubuntu.com/
26http://fedoraproject.org/wiki/Extras/UsingExtras
27http://www.debian.org/doc/manuals/apt-howto/ch-basico.en.html#s-sources.list
29http://www.informatik.uni-koeln.de/fai/
run multiple times for all updates to finish. Neither Apple nor Microsoft allow third party packages to be managed via their updater systems; in fact, though Apple typically updates their after-market software along with the operating system via Software Updater.app, Microsoft maintains separate update systems for Windows and its popular Office application suite.

With the emphasis on application deployment, it is difficult to see how managed package repositories can aid in maintenance of user data.

3 What’s missing?

All of the administrative tools shown above aid workers in application administration, but none of them focus on the other (arguably more important) half of the modern computing environment: data. In the spirit of ‘the simplest possible thing that could work’, administrators often set up a network mountpoint with NFS, SMB/CIFS, or other network-mountable file systems. These online bit buckets are simple to create, but require that the user always remember, when saving, to navigate to the shared mountpoint, or risk leaving his work only on the current workstation. FTP, its cousins (ssh’s remote copying and FTP-clone methods) and mirroring tools such as rsync don’t require a continuous connection, but do require even more thought on the part of the user.

With Apple’s Open Directory server on Mac OS X 10.4, users’ home directories can be cached using the Portable Home Directories feature. Portable Home Directories are an interesting model for solving the data problem: on Mac OS X, the only place most users can save files is in their home directory, which is synchronized upon login and logout, or manually (if the user’s home directory is too large to practically update every session). Although it has some implementation problems — most glaringly, its default omission of the user’s configuration information in ~/Library — Portable Home Directories go a long way towards solving the problem on the Mac, and providing a case study for more cross-platform attempts. Microsoft’s Roaming User Profiles allow similar functionality, in that documents and settings move with a user around the network; however, they do not seem to be able to function in a disconnected state as Portable Home Directories do.

3.1 Model, View, Controller

Whether under the name of Model/View/Controller (hereafter MVC) or Presentation/Abstraction/Control, large computing systems have lent themselves...
to a common, tripartite division. The traditional example involves a desktop graphical user interface program; it will contain user interface views, data models to describe the application-specific problems, and controllers to facilitate communication between models and controllers. From Apple’s most rudimentary developer examples\textsuperscript{36} to their higher-level conceptual documentation\textsuperscript{37}, MVC frequently arises as a strong aid for maintainable, reliable programming. Sun Microsystems’ developer documentation calls MVC the “recommended architectural design pattern for interactive applications”.\textsuperscript{38}

The separation provided by MVC avoids unnecessary duplication of work. A poorly coded web application that ignores MVC will require a programmer to deal with business logic and application-specific behaviors when only presentation needs to be changed. Likewise, a fragile application with interface details and application functions improperly separated will mean user interface elements are tied down to one specific concept; this reduces reusability and makes application consistency harder to obtain. With a proper MVC design, new models can be introduced without perturbing the interface; likewise, changes in the presentation layer will not introduce complications in a program’s core functionality.

The central concept of the Stateless implementation at hand is that MVC can solve the same problems for computer administrators as it does for programmers. Current administrative practices suffer the same deficiencies as non-MVC applications; namely, they conflate user data and applications, if they even bother to consider user data in the first place. A well-run organization can parallel a well-designed program in many respects:

<table>
<thead>
<tr>
<th>Computer</th>
<th>Organization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>Data, state information</td>
</tr>
<tr>
<td>View</td>
<td>Buttons, text fields, web sites</td>
</tr>
<tr>
<td>Controller</td>
<td>Input processor</td>
</tr>
</tbody>
</table>

With these parallels in mind, it seems reasonable to suggest that a MVC approach to system administration might afford the same advantages as the MVC division of software internals. In such an environment, ‘model’ information would never be tied to a specific ‘view’ — that is to say, no data or user configuration would reside solely on one workstation. Much like the software approach, administrators would be free to add new models (‘state information’ such as users, documents or application preferences) or views (new workstations, terminals, laptops, or web applications, or services) with little to no interruption in the remainder of the organization. Active Directory, which uses a central

\textsuperscript{36}http://developer.apple.com/documentation/Cocoa/Conceptual/ObjCTutorial/chapter02/chapter2_section3.html


\textsuperscript{38}http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/web-tier/web-tier5.html
administrative viewpoint to manage computers and users separately, is moving in this direction, but is not explicitly MVC and does not explore the benefits of such an arrangement.

4 Base technologies

The initial version of Stateless, targeted at Fedora Core Linux with an eye to near-immediate Mac OS X portability, will build off the following technologies:

1. **LDAP**: all workstations will authenticate against an OpenLDAP server, ensuring that a user with an account on one machine has an account on all machines of that class; additionally, the user’s record in will contain a reference to the user’s online state information storage. The LDAP server will also maintain a record for each class of computers, detailing where the latest snapshot of that class can be found, and what the latest revision number is. Workstations will poll the LDAP server hourly to check for new updates, and apply them in the background.\(^\text{39}\)

2. **rsync**: Initially, transferring snapshot updates will take place over rsync to ease network congestion and take advantage of the already existing command-line tools and libraries. Over time, rsync may become suboptimal for this task; in which case, rsync functionality should be encapsulated by a plugin system, such that new updating systems can be easily snapped in on request. To keep from interrupting a user and from allowing incomplete updates, the system would follow Havoc Pennington’s advice of using two partitions – a Live and Reserve. Only the Reserve partition would be updated; upon a successful update, the computer would use the next reboot to swap the Reserve and Live partitions, bringing the user access to the updated software. \(^\text{40}\) Rsync may also be used to transfer fresh state information onto a workstation from the user’s central storage, and to upload new state information to the network storage after a session.

3. **yum**: The central image machine will be maintained as though it were a single workstation in a traditional environment, which is to say, all packages will be installed via .rpm files from a yum repository. This provides the usual package management benefits (easy installation and removal).

Certain tools will have to be written to tie the preexisting technologies together:

1. **Snapshooter**: The snapshooter will perform any standard cleaning tasks on a standalone system before committing it to network storage and incrementing the snapshot revision kept in LDAP. Its tasks might include removing local users and home directories, clearing out temporary caches,\

\(^{39}\)Polling is generally considered a poor design decision, but it enjoys use in apt, the Mac’s Software Update, Windows Update, and yum

\(^{40}\)http://people.redhat.com/ hp/stateless/StatelessLinux.pdf, page 2
or running a series of administrator-supplied scripts to test that necessary
functionality such as LDAP authentication and end-user network mount-
ing are enabled.

2. Updater: The updater will periodically check the LDAP server to ensure
that the workstation has updated to the latest requested snapshot, and
apply new snapshots to the system without interrupting the end-user.

3. Central management tool: as a wrapper around LDAP calls, the man-
agement tool would allow administrators to roll a class of machines back
to a specific snapshot, or change the class that any number of worksta-
tions belongs to. Additionally, through the management tool adminis-
trators could associate groups of users with certain classes of machines —
for instance, limiting developer logins to all machines falling in the
developer-workstation class of computers, but allowing both developers
and salespersons access to the demo-workstation class. Management tools
from Apple, Microsoft, Novell and others already perform similar tasks;
at the very least, their interfaces can be useful guideposts, but hopefully
Stateless management tools can be integrated using preexisting manage-
ment tool’s plugin interfaces (such as Active Directory snap-ins\(^{41}\) or Open
Directory plug-ins\(^ {42} \)).

5 Pitfalls: Or, What Stateless is Not

To avoid the problems of the previously mentioned systems, Stateless must never
allow the following to be true:

1. Lack of network access means users can’t work. To avoid this problem,
a Stateless client will have to cache data aggressively. However, network
based logins (for instance, stored in LDAP) will mean users cannot log into
disconnected computers. A number of measures can be used to combat this
problem; similar to Apple’s OS X, a directory service could run locally\(^ {43}\),
but if this approach proves to be too unmaintainable or heavyweight,
authentication modules such as pam\(_{ccreds}\)^{44} hold promise.

2. Stateless only works on one platform.

3. Stateless updates require extensive downtime. or Stateless updates perform
redundant, needless work By using rsync or rdiff\(^ {45} \) to perform exact,
differential binary updates, updates will take the minimum amount of
time with the maximum amount of consistency. By adopting the Fedora

\(^{41}\)http://msdn.microsoft.com/library/default.asp?url=/library/en-
us/mmc/mmc/microsoft_management_console_start_page.asp
\(^{42}\)http://developer.apple.com/networking/opendirectory.html
netinfod.8.html#/apple_ref/doc/man/8/netinfod
\(^{44}\)http://www.padl.com/OSS/pam_ccreds.html
\(^{45}\)http://www.nongnu.org/rdiff-backup/
Stateless concept of Live and Reserve partitions, updates can be applied nearly transparently and with no downtime.

4. *Interrupted Stateless updates ruin the installation.*