

# About the Integration of Mac OS X Devices into a Centrally Managed UNIX Environment

Anton Schultschik – ETH, Zurich, Switzerland

## ABSTRACT

The UNIX flavors in use today have so much in common that centralized management of UNIX systems has become almost standard. Since Mac OS X is based on BSD-UNIX it is a promising candidate for integration into a centrally managed UNIX environment.

Starting from generic administration concepts, this paper develops an integrated management concept that handles fully automated installation and configuration of hosts. The concept includes a centralized application management system for console and graphical Mac OS X applications.

The management concept is then implemented based exclusively on standard UNIX tools. The necessary extensions of these tools to make Mac OS X conform to UNIX standards are presented, including a proxy tool to forward AppleEvents which facilitate the interprocess communication for centrally managed graphical Mac OS X applications.

## Introduction

The increasing diversity of hardware and software makes system management more difficult. Shorter life-cycles of computer systems require more frequent upgrades or replacement of hardware and as a consequence, the installed computers on a large site rarely are uniform in hardware but rather split into several uniform clusters. Automated management of such an environment is challenging as complexity grows with each new configuration of hardware and software.

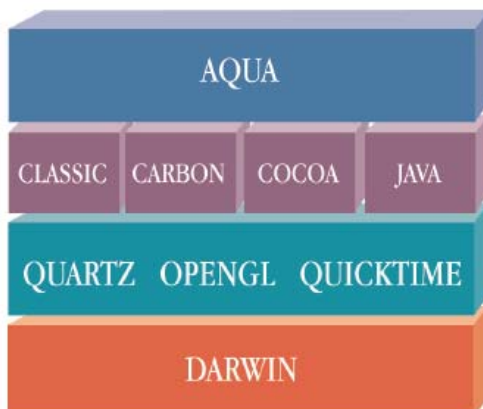


Figure 1: Structure of the Mac OS X operating system (from [2]).

Through integrated management of the clusters common tools and common configuration information can be reused across the clusters thus reducing the amount of information required to fully understand the entire site. With Mac OS X being a member of the UNIX family integrated management of Mac OS X in a UNIX environment comes within reach. In this paper, design,

implementation and deployment of such a system will be layed out built on some basic principles. The result will be a management system for UNIX systems that can be used to manage network-based as well as stand-alone systems.

In system management three basic principles keep appearing in tools and methods [1], and these shall be used as orientation for our integrated management system:

- *Reproducibility* ensures that the same action produces identical results. Automation, a way to implement reproducibility, helps to exclude human error in repetitive tasks.
- *Comprehensibility* of all actions is necessary for the administrator especially when troubleshooting or modifying the configuration.
- *Avoidance of Redundancy* helps to keep data consistent and thus easier to manage.

## Mac OS X is UNIX Plus . . .

Mac OS X unites the strengths in UI and application design of previous Macintosh operating systems with the stability and flexibility of a modern UNIX platform. Integrating Mac OS X into a general UNIX environment requires a closer look at the operating system since not all parts in Mac OS X have their origin in the UNIX world:

- **Darwin:** Darwin is based on Free-BSD including the standard UNIX network clients and servers as well as the usual user space utilities. The appearance of some daemons and configuration files have been modified to match with the rest of Mac OS X.
- **Quartz, OpenGL, QuickTime:** Instead of relying on the UNIX X11 standard, Apple

decided to build an alternative graphic system.

- **Classic:** The Classic environment provides emulation support for native pre-OSX applications. These applications only work with Apple's HFS/HFS+ filesystem.
- **Carbon:** The Carbon library framework provides compatibility to pre-OSX system calls at source code level. Carbonized applications also run on non-HFS filesystems through Carbon's HFS emulation although with reduced stability.
- **Cocoa:** Providing an entire new standard for application development, Cocoa is based on modern, UNIX compatible technologies like XML and Java.
- **Aqua:** The top layer of Figure 1 represents the graphical user interface on which the different GUI applications run.

Looking at the entire operating system, the Darwin roots as well as modern Cocoa-based applications are fully compatible with the rest of the UNIX world even though Apple did not use the X11 standards in their graphics system. Thus the key challenge in managing a Mac OS X system as a UNIX is the handling of legacy applications and their specialties.

#### Review of Available Tools

Several system management tools are available under Mac OS X that focus on three different management areas:

- **Installation** of the operating system
- **Configuration** of operating system and applications
- **Software distribution** or installation onto an installed host

The management tools need to be applicable on classic UNIX flavors while supporting the Mac OS X specific extensions, e.g., legacy application support. Several candidate tools were considered, and their strengths and weaknesses will be discussed in the following sections.

#### Installation Tools

Several strategies can be chosen to install an operating system onto a target host. Network-based installation allows access to centralized services and is logistically efficient. Therefore only network-based installer methods are considered in the choice of tools.

##### *NetInstall*

Apple provides a native mechanism [3] to install multiple client machines based on the installation of packages (.pkg bundles [4]). The target host of an installation is net-booted from a modified disk image that will start an installer system. The installer then installs packages supplied on the booted image.

The package-based approach of NetInstall yields reproducible and certainly comprehensible results since all changes on the installed system have their source in one of the installed packages. As the installer image can only contain a single configuration several NetInstall

images are required when managing multiple host configurations. Consequently, each image would redundantly contain commonly installed packages making the management of NetInstall a difficult task in a heterogeneous environment.

##### *Net-Restore*

Mike Bombich's NetRestore [5] is a suite of GUI tools that are based on ASR, Apple's image management tool [6]. An installation is started by net-booting a target host into the NetRestore installer in which the administrator selects the image to be restored to the local disk. The individual images are supplied through a network share along with post-installation scripts for the individual configuration of the installed host.

Since the restored system is identical with the source image the installation itself clearly is reproducible. However creation and maintenance of the source image is done by hand and the final system can not be comprehensible as a whole. An administrator neither explicitly sees why a system is in the current state nor completely understands the consequences of each manual step during assembly. As with NetInstall in the previous section the management of several configurations implies the use of multiple disk images introducing redundancy between the manually maintained images.

##### *Sun Solaris Jumpstart*

Network based installation is done by net-booting an installation target into the Sun Solaris Jumpstart [7] system. Once booted Jumpstart uses DHCP and DNS to determine the correct configuration list of packages and appropriate pre/post-processing scripts.

The Jumpstart configuration concept is simple and yet capable of comprehensibly handling individual configurations. Its design for completely unattended installation makes the Jumpstart system reproducible.

#### Configuration Tools

To reproducibly maintain the configuration of systems, especially in a heterogeneous environment, automated tools are essential. However to provide the necessary comprehensibility, configuration information consisting of a large number of modifications for a target system must be structured into modules. By postulating module integrity, i.e., that no module destroys the modifications of another, reuse of modules becomes possible, thus controlling unwanted redundancy. The following three tools fulfill this basic requirement.

##### *Cfengine*

One of the best known tools comes from Mark Burgess of Oslo University College. Cfengine [8] is a highly flexible scripting system that deducts its configuration based on the context of a managed host. Cfengine supports various UNIX flavors including Mac OS X and is equipped with its own file sharing mechanism.

Provided that Cfengine is run in the same context, reproducible results can be expected, and modularization is provided through the *classes* construct.

However, since Cfengine does not enforce integrity of actions or classes, configuration scripts can easily exceed the state of comprehensibility.

#### **Radmind**

Radmind [9] is available on various UNIX dialects including Mac OS X. Designed for ease of use, Radmind implements a capture and replay strategy resulting in a tree of dependent *load sets* consisting of files modified during a capture session.

The capture and replay processes alone are comprehensible and reproducible. However the dependencies between load sets restrict the replay of features between different lines of history. As these restrictions are not managed by Radmind the replay of an unsuitable load set impairs the reproducibility of functionality and the comprehensibility of the configuration.

#### **Template Tree 2**

Template Tree 2 (TeTre2) [10] is used for the administration of UNIX clusters. The configuration of a system is managed using simple file operations structured into integrity-preserving modules. A subset of Cfengine functionality is used to propagate the configuration to a target system.

Although dependencies between features in a TeTre2 configuration exist, these dependencies are functional rather than historical. This allows features to be recombined freely without impact on comprehensibility or reproducibility.

#### **Software-Distribution Methods**

When managing software, the integrity of application packages and the operating system ensure that all available applications are functioning. Knowing the origin of each file in the system is the basis to maintain such integrity. Especially when software packages are installed intrusively, i.e., by copying them into the standard directory tree of a system, the tracking of file origins becomes difficult and thus needs to be examined more closely.

#### **Fink and the Debian Packaging System**

Software built within the Fink Project [11] is packaged in Debian packages, that are installed with a well supported tool-suite. A database keeps track of the origin of installed files by associating them with their source package. An installer then relies on this database to ensure the package integrity.

Software management using Debian packages is thus reproducible and comprehensible. In a larger environment where multiple versions of the same package need to exist file collisions can only be resolved through renaming or versioning, e.g., gcc-2.95 and gcc-3.3. As not all files are easily relocatable and the resulting versioned structure always enforces version dependencies, creation and administration of such packages would be expensive.

#### **Network-based Distribution Via SEPP**

The SEPP-Packaging [12] system follows a different concept to distribute applications. Rather than

installing applications file by file into an existing installation each *SEPP-package* encapsulates a ready-to-use application within a separate directory. Once this package directory has been copied or mounted over NFS the packaged applications are made accessible to the user using stub scripts. By design SEPP supports the coexistence of multiple versions even when dependencies on other SEPP-packages exist.

Through its structure a SEPP installation is comprehensible and yields reproducible results. In addition the Mac OS X standard proposes a similar approach to encapsulate the files of an application into a bundle directory. Thus incorporating Mac OS X support into SEPP is straight forward.

### **A System Management Concept for Mac OS X**

#### **Available Infrastructure**

When implementing our Mac OS X management concept, we relied on the existing infrastructure at our site. This infrastructure consists of the provided network services and the available tools. These two aspects of the infrastructure are discussed in this section.

#### **Available UNIX Network Services**

Centralization of services is a common approach to reduce redundancy of information. In the implementation of the integration concept the following services were used:

- **DHCP:** Management of network configuration and access control
- **LDAP:** Consistent user authentication and group management
- **NFSv3:** Network-based user homes and application packages
- **SMB:** External access to user homes

While all other services are only available using a single protocol to prevent confusion, user homes can be accessed using two different services: NFS and SMB. NFS is used as default for reasons of speed and flexibility. However the current implementation of NFS requires a trusted network and accordingly SMB is used for remote access from home or across untrusted networks.

#### **Chosen Tools**

Several administration tools were introduced earlier, each being particularly strong in one of the administration areas. This information is now used to determine the best suitable tools to implement an integrated concept:

- **Network Installation of Mac OS X:** Jumpstart is the best choice even though reimplementa-tion is necessary because it runs fully automatic and completely unattended.
- **Configuring the System:** The built-in limitation of complexity as well as the support for modularization make TeTre2 the preferred tool for a structured approach to integration.

- **Serving Applications:** Because the SEPP approach matches with the Mac OS X application concept SEPP is an ideal choice even though an extension for graphical applications needs to be implemented.

In the next sections we first introduce each of the tools in its original form followed by a description of the changes necessary to make it work on Mac OS X.

**Network Installation of Mac OS X Based On UNIX Standard Protocols**

*Design of Solaris Jumpstart*

Jumpstart is based on the standard UNIX services BOOTP/DHCP, TFTP and NFS. Once it is net-booted, it performs an unattended installation as described in Figure 2.

The jumpstart system is net-booted on the target host. Therefore the DNS name of the target host is known to jumpstart at execution time and thus can be used to access a configuration directory on an NFS share with the name of the target host. This directory contains the pre-/post-installation scripts and a list of packages that shall be installed.

*The Reimplementation of Jumpstart for Mac OS X*

Our Mac OS X reimplementation of Solaris Jumpstart provides the same basic functionality and several useful extensions. Some of these extensions are listed in gray color in Figure 2.

- **DHCP:** Although Darwin is compatible with UNIX standards, the net-boot implementation of Macintosh open-firmware is incompatible with the normal DHCP protocol and tries to start negotiations with the DHCP server. With a patch found in [13] it was possible to net-boot the installer using Linux and Sun Solaris.
- **Override for hostname:** One configuration file in TeTre2 associates all host names with their Ethernet MAC addresses. With the information from this file osxjumpstart can set the correct host name even when a host is booted of a temporary IP address and choose a different host

name for the configuration. We use this feature when installing portable Macs from a common set of “install only” IP addresses.

- **OS Installation:** Since applications are supplied using SEPP, only the operating system and extensions, e.g., fonts, are installed via osxjumpstart. The OS packages are provided in their native .pkg bundle form [4, 14]. Additional self-built packages are also provided in this format for consistency.
- **Software Update:** Using Apple’s softwareupdate tool Mac OS X and installed packages are brought up-to-date. Unfortunately softwareupdate was designed to update an already running system. Thus the tool is run in a chrooted environment on the newly installed system without requiring a reboot.
- **Management of Sensitive Data:** All sensitive data, e.g., passwords, licenses, . . . , is stored in an encrypted archive and the administrator is required to enter the encryption key at the beginning of the installation. In the post-install actions sensitive data is then copied into the system.

Passwords for users in the netinfo database are stored in separate files under /var/db/shadow/hash. The hash files can be copied to the archive from any Mac OS X system with appropriately set passwords. The same strategy can also be applied to the Auto-login password (/etc/kcpassword) and the Open Firmware password (nvram security-password and nvram security-mode).

**Configuring the System**

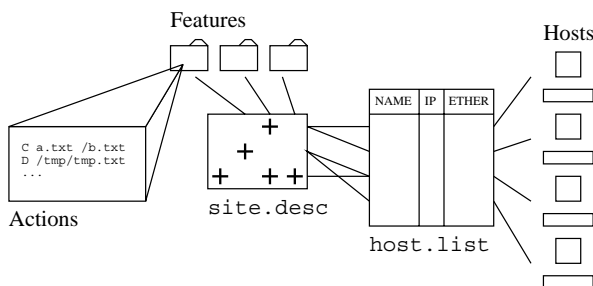
*Design of Template Tree 2*

Template Tree 2 (TeTre2) is a configuration management system that structures the configuration of an entire site into modules – *features* in TeTre2 terminology. The configuration of a host is defined as a set of features whereas features consist of simple file operations. Figure 3 shows the structure of TeTre2. Basic information like hardware type or Ethernet MAC

Net-Boot	<ul style="list-style-type: none"> <li>• Determine IP using DHCP or BOOTP</li> <li>• Boot the installer system from TFTP, NFS</li> <li>• Mount required NFS shares</li> </ul>
Config-retrieval	<ul style="list-style-type: none"> <li>• Get host name from IP-number and DNS</li> <li>• Override host name for specified Ethernet address</li> <li>• Use hostname to retrieve config info from NFS</li> </ul>
Pre-Install	<ul style="list-style-type: none"> <li>• Format/partition disks</li> </ul>
OS/SW installation	<ul style="list-style-type: none"> <li>• Install required packages</li> <li>• Update the system using softwareupdate</li> </ul>
Post-Install	<ul style="list-style-type: none"> <li>• Configure the system (TeTre2)</li> <li>• Transfer sensitive data</li> <li>• Optional custom actions</li> </ul>
Reboot	<ul style="list-style-type: none"> <li>• Start the installed system</li> </ul>

Figure 2: Steps of a jumpstart installation (black) and the extensions for osxjumpstart (gray).

address of each host (the `host.list` file) on a site is separated from the individual host configuration for all managed hosts (file `site.desc`).



**Figure 3:** Structure of the TeTre2 Configuration: In the `host.list` a hostname is associated with basic configuration information like installed OS, Ethernet Address or Network configuration that can be used as text substitutes in configuration actions. The `site.desc` file then selects the features for a host. Finally each feature contains a file containing a list of configuration actions that are applied to a host if the associated feature has been selected in `site.desc`.

Features encapsulate a certain functionality, e.g., install and activate a service, or behavior, e.g., power management. The ability to enforce integrity allows flexible combination between features and avoids coincidental dependencies. Dependencies between features are designed by the administrator. As a consequence TeTre2 features obtain a semantic aspect in the context of system administration.

Looking inside a feature one will find simple file operations such as copying files, creating and removing directories and generating symlinks. The assembly of files from chunks, simple text substitution and operating system dependent execution of file operations provide more flexibility for the configuration work. For the sake of comprehensibility TeTre2 never modifies the content of files. Either the files are fully controlled and consequently overwritten or the files are outside the scope of TeTre2 and thus ignored. Files with mixed content, e.g., that partially depend on TeTre2 configuration are managed by custom scripts running on the target system. For example allowing users to add their own printers and still manage a default set of printers through TeTre2 requires this mixed form of control.

#### Overview Over the Mac OS X TeTre2 Features

The functionality of the TeTre2 tool itself was sufficient to configure Mac OS X systems without modification. However a multitude of new features were required to control the diverse aspects of a Mac OS X installation. Figure 4 gives an overview of the implemented Mac OS X features. The categorization of the features is not part of TeTre2 but was introduced in the table for the benefit of the reader.

<i>Basics:</i>	Automounter, Network configuration, Netinfo builder
<i>Admin tools:</i>	SSH public-key access, automatic package installer
<i>Services:</i>	Postfix, sshd, Filemaker/Meetingmaker server
<i>Clients:</i>	LDAP access, automount NFS homes, SEPP
<i>User specific:</i>	Default printers, custom loginwindow dialog

**Figure 4:** Categories of TeTre2 features with examples.

Unless documentation is available, features are created by identifying and verifying the differences that occur when settings are modified through the Mac OS X GUI. The concerned files are stored in the feature and copied during the configuration process. If several features contribute to the content of a file, chunk-wise assembly is the solution. However when the content of a file depends on the current state of the system, e.g., hardware, the files are generated through scripts and the configuration for these scripts as well as the installation of the scripts are handled with TeTre2. Some of the features we added to TeTre2 for Mac OS X support were:

- **Package Installer** Adapted from the corresponding UNIX feature an automatic post-install .pkg bundle installation is performed through a script. The package installer is run at each system start as StartupItem and is also run nightly by cron. Packages missing on the local system get installed based on a configuration directory. Upgrades of existing packages and required reboots are handled, the latter is done through the reboot command or by sending an AppleEvent to the Finder when a user is logged in.
- **Netinfo Database Builder** Netinfo data is stored in binary format making direct control through TeTre2 impossible. As the Netinfo database is required by Mac OS X for some applications parts of Netinfo are regenerated at each start using the Netinfo Builder. Netinfo Builder is a Perl script that compares the current state of the Database with a desired state in the form of a raw dump (the term *raw* might be imprecise since the dump is ASCII but in a general format). Netinfo Builder only changes the differing elements allowing a coexistence between managed and individual settings.

#### Serving Mac OS X Applications from NFS

##### Design of SEPP for Command-line Applications

The design goal behind the SEPP [12] application distribution system is the encapsulation of ready-to-run application packages into individual directories. As a consequence the application executables inside a SEPP package need to be *reconnected* to the OS and its user. SEPP is structured into two parts that are

separated physically in two directory trees as shown in Figure 5. The application packages are located under `/usr/pack/...` Optional support for NFS automount is built into SEPP allowing application packages to be assembled from various sources.

The `/usr/sepp/` directory is used to manage the applications on the local system making them available through the `/bin/...` directory. Rather than symlinking the executables from an application package, a Perl script – a stub – stands for the real application binary. Upon execution the Perl stub (in `/usr/sepp/bin/`) starts the real executable inside the package triggering the automount of the package containing the application. More precisely the Perl stub starts a package specific start script named `start.pl` (inside each `/usr/pack/...`) which then can properly set everything up before starting the application. The concentration of all administration information into one directory (`/usr/sepp/`) makes it possible to distribute this directory via NFS resulting in an application system that is fully centralized.

**SEPP for Mac OS X**

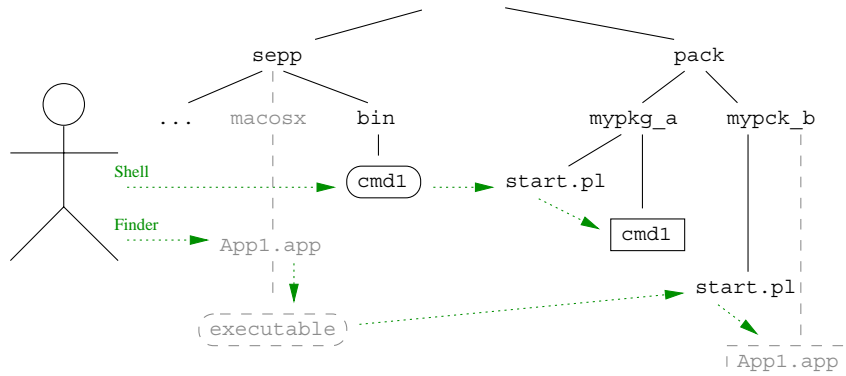
SEPP supports Darwin and X11 applications on Mac OS X in the unmodified form. In the case of Cocoa or Carbon applications the basic structure of Mac OS X applications [15] comes into play: Mac OS X applications are packaged into a single, structured

`.app` bundle directory containing application data or additional libraries required by the application. Usually the files and directories within the bundle are addressed by relative paths allowing an `.app` bundle to be moved around on the system without re-installation.

Making the packaged applications available to the Finder is more of a challenge. A closer look at the anatomy of a `.app` bundle however leads to the following solution. The Finder only needs a part of the bundle files to display the application and associated documents. We use this fact in the Mac OS X extensions of SEPP by copying all the required files to form an independent miniature application – an *application stub* – in the `/usr/sepp/macosx/` directory. In this application stub a small Perl script is used to start the real application. Figure 6 shows the structure of such an application stub directory. By making the contents of the `/usr/sepp/macosx` directory available as an item in the Finder’s left hand navigation bar users can access all SEPP applications in a natural Mac-like way.

**Special Application Support**

All binaries in a SEPP package are started indirectly through the package specific `start.pl` script. One of the advantages of such a script is the ability to prepare the environment for an application. Mac OS X SEPP packages make regular use of `start.pl` since many applications do not follow the standard exactly.



**Figure 5:** Structure of the SEPP system: The original SEPP system designed for command-line applications is shown in black and the extensions implemented to support Mac OS X GUI applications are shown in gray. The boxes in the figure represent executables. Rounded boxes are used for the Perl stubs that a user will execute directly while the cornered boxes stand for the real applications. The individual `start.pl` files found in every package provide customization for the application to be started. The dotted arrows show the chain of actions when an application is launched.

```

... ./Sample.app/
  Contents/
    Info.plist          Contains filenames of executable and icons
  MacOS/
    the_executable     Perl script replacing the real executable
  Resources/
    sample_icon1.icns  Displayed in Finder
    sample_icon2.icns
    ...icns            All other Icons referenced in Info.plist
    
```

**Figure 6:** Structure of a Mini application bundle used as application stub. Entry point is the `Info.plist` file in which all other filenames of the bundle are found.

Providing locally installed Files to Applications  
Some Mac OS X applications require files to be at a certain location in the system. Often the current user's permissions are not sufficient and as a consequence these modifications cannot be done in `start.pl`. Therefore a `sudo` gateway has been opened for SEPP packages under Mac OS X allowing `start.pl` to run a second package specific script with administrator privileges.

To prevent abuse of this mechanism `sudo` is restricted to a small SEPP tool which starts the script after ensuring that the script is at a valid location inside the package with correct ownership and permissions. Therefore only the package maintainer can create privileged scripts that are allowed to run.

Legacy Carbon Applications Many commercial applications are still running under Carbon and depend on a HFS-like filesystem behavior. While many Carbon applications remain fully functional when started from NFS, some applications insist on being stored in HFS volumes.

To support these special Carbon applications they are transferred onto a HFS formatted disk image that is stored in the SEPP package on any filesystem since the disk images are managed by Darwin. `start.pl` mounts this image and executes the application on HFS. Using shadow file support these images even can be made writable.

#### Forwarding AppleEvents With `evntproxy`

AppleEvents [16] provide a convenient way for applications to communicate among each other. This includes the ability of one application accessing another application to open or print a document for example.

Under SEPP an application stub is started before the real application and in some cases this indirect start can cause AppleEvents from Finder to be sent to the stub instead of to the real application. Consequently the real application cannot participate in AppleEvent interactions

We have solved this problem with the creation of a proxy for AppleEvents – `evntproxy` – which is started in the application stub. Figure 7 shows the sequence diagram for the handling of AppleEvents through `evntproxy`.

#### Experience Gathered on the Deployed System

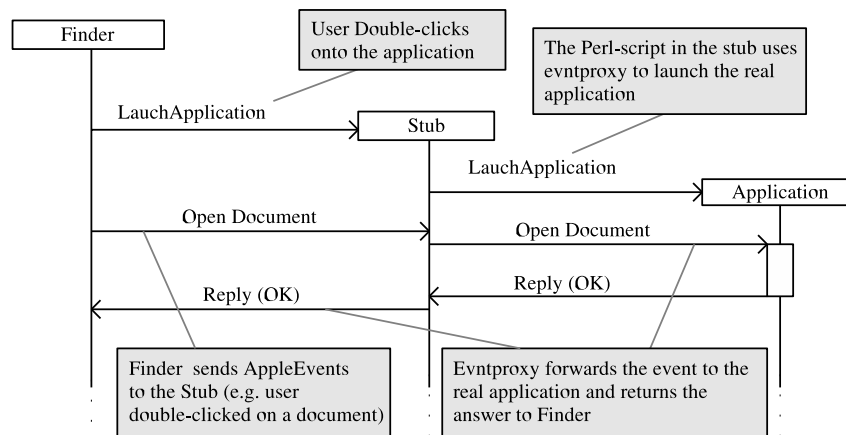
##### Current Installation

At the time of writing the integrated Mac OS X system described here is installed on more than 50 computers consisting of two homogeneous clusters as well as several individual computers with various G4 and G5 hardware. More than 80 SEPP application packages exist covering the frequently used commercial applications from Microsoft, Adobe and Apple, as well as a variety of UNIX tools and applications. Centralized infrastructure is provided by two Sun Solaris and one Linux server. Special application services like Meetingmaker and Filemaker server are run from an Apple XServe that is also managed through `osxjumpstart`, `TeTre2` and SEPP.

##### Migration & Maintenance

Since most of the integrated Macs are connected to the same local network and thus to the same services only a small subset of features required customization. As `TeTre2` is able to activate individual features for a predefined group of managed hosts, the configuration of the two clusters could be created in a very short time. Even for the individually configured Macs many features could be reused and thus applied group-wise.

The installation was done at the users work place with no need to transport any hardware. As all installation resources are network based simultaneous installation of several Macs was possible. In some cases only untrained teaching assistants equipped with the necessary passwords were required on site since `jumpstart` is managed centrally and runs unattended. Public-key SSH access during installation allowed an administrator to intervene remotely when required.



**Figure 7:** Sequence diagram of AppleEvents exchanged between the Finder, the stub for an application, and the application itself. After being started by the Finder the stub itself starts the real application using `evntproxy`. `Evntproxy` then forwards all AppleEvents to and from the real application.

Since an entire `osxjumpstart` installation takes about 30 minutes without requiring the presence of an administrator, quicker reaction is possible when replacing defective hardware, e.g., to set up a replacement Mac.

The automounter under Mac OS X Panther presented us with a number of challenges: When reloading larger automount-maps, the automount shares were not available for the fraction of a second. The reason for this is that the automounter removes and regenerates all the symlinks it puts in place to trigger automounts as soon as it has to reload its maps. A second instability in the automounter allowed the mount directory to be accessed before the mount was completed. As a result some files get accessed on the local directory rather than the NFS network shares. To ensure the availability of the users home directories a script is run in the `login_hook` of the LoginWindow application, triggering the mount and then waiting for its completion. A first look at the automounter in Mac OS X Tiger revealed several modifications raising the hopes that an OS upgrade will provide a more stable automounter.

### User Acceptance

The introduction of the integrated Mac OS X concept was widely accepted on the user side. From a user's point of view the integrated system looks almost identical to a normal Mac OS X environment and users familiar with Mac OS X observe the following changes:

- The users are able to roam between workstations and continue to work in an identical setup since user homes and applications are managed centrally.
- Applications provided by SEPP are found under `/Network/Applications`. As multiple versions of a package may exist the recommended version is accessible through the subdirectory `Default`.
- The start of an application is slightly slower as it is transferred over the network. This effect occurs only during start. Once the application is cached the user is no longer slowed down and can work at normal speed.
- Since SEPP is managed centrally, newly installed applications appear on all managed Mac OS X hosts without any user interaction.
- The homes of *all* other users are available through NFS including UNIX and Windows users. Consequently the integrated Mac-Users need to take more care with file permissions.
- A check for pending software updates is done each night or in a `StartupItem` during boot. If a reboot is required for an update, the user is notified and given the chance to save all documents.
- The NFS based home directories are automatically backed up every night which allows to restore files that were accidentally overwritten or deleted.

### Limitations

So far all of the encountered applications could be brought into a form that makes them work in a SEPP

environment. However the effort to do so is sometimes not justified. In the case where an application is only used on one or two hosts or when an application is too closely connected to the operating system (e.g., Scanner-Software, some server applications) it is more efficient to capture the application into a `.pkg` bundle and install it with `osxjumpstart`.

### Special Configurations

The integrated Mac OS X system was also deployed on five Macs which are only connected to the servers via an insecure, low-bandwidth network. NFS could not be used either during installation or operation. By leaving aside the TeTre2 features for network services and by implementing the necessary features to substitute required functionality the configuration for a stand-alone Mac was created.

SEPP packages are installed on the local disk with the option to `rsync` and install other packages. User homes are stored locally. To allow these users to roam their data is synchronized with their network home regularly using `unison`.

To support on site installation of the stand-alone Mac OS X systems a disk-based version of `osxjumpstart` has been created that works independent of network resources. The copied `.pkg` bundles, SEPP packages, and the `osxjumpstart` and TeTre2 configuration has to be updated from the network based master copy regularly.

### Conclusion

A system management concept has been implemented that allows us to manage Apple's Mac OS X as part of an integrated UNIX environment. Fully automated administration tools are provided that support fully unattended and highly flexible system installation and configuration based on common UNIX standards and a centralized network infrastructure. A software management tool allows the users to work transparently with applications residing on multiple local and network-based sources without prior installation.

As the same administration structures are used for several software platforms fewer tools need to be mastered and maintained and an administrator can proceed identically when working on different systems. The reuse of configuration information is supported by the tools even across software platforms. As the tools are designed for limited complexity an administrator is able to understand the consequences of his or her actions more easily.

The integrated Mac OS X system shows only subtle changes versus a vanilla Mac OS X system and thus allows users to orient themselves easily. Due to the modular structure of the configuration data and its automated application, state and behavior of a system is intuitively predictable increasing the user's confidence in the installation. The option to roam between workstations allows more flexibility to the user while reducing the impact of hardware failure.

Reproducibility, comprehensibility and centralization require some effort to implement but in the long run this investment is paid back both through reduced maintenance cost and higher user satisfaction.

#### Availability

All code for the integrated Mac OS X system is licensed under the terms of the GNU GPL. At the time of writing the SEPP extensions for Mac OS X already are available under <http://isg.ee.ethz.ch/tools>. The remaining code is to be published on the same URL.

#### Author Biography

After having developed image processing software in both an industrial and a scientific environment, Anton Schultschik found his way to system administration as member of the IT support group (ISG.EE) at the Department of Information Technology and Electrical Engineering at the Swiss Federal Institute of Technology (ETH Zurich). With a focus on Mac and Solaris/Linux support and development, he also maintains the local LDAP directory services and the Condor batch processing cluster. In his spare time he likes to spend time with his newborn son or extend his knowledge on various IT topics.

#### Bibliography

- [1] Traugott, S. and J. Huddleston, "Bootstrapping an Infrastructure," *Proceedings of the 12th Systems Administration Conference (LISA)*, 1998.
- [2] Apple, *Macosx, An Overview for Developers*, [http://www.tri.ucalgary.ca/tri1/Downloads/Mac/OSXDeveloper/macosx\\_overview.pdf](http://www.tri.ucalgary.ca/tri1/Downloads/Mac/OSXDeveloper/macosx_overview.pdf).
- [3] Apple Developer Connection, *Mac OSX Server System Imaging and Software Update Administration*, [http://images.apple.com/server/pdfs/System\\_Image\\_and\\_SW\\_Update\\_v10.4.pdf](http://images.apple.com/server/pdfs/System_Image_and_SW_Update_v10.4.pdf).
- [4] Apple Developer Connection, *Introduction to Software Distribution*, <http://developer.apple.com/documentation/DeveloperTools/Conceptual/SoftwareDistribution/index.html>.
- [5] Bombich, Mike, *Bombich Software: NetRestore*, <http://www.bombich.com/software/netrestore.html>.
- [6] Apple, *ASR – Apple Software Restore*, <http://developer.apple.com/documentation/Darwin/Reference/ManPages/man8/asr.8.html>.
- [7] Amarin, Kevin, *Solaris Jumpstart Automated Installation*, <http://www.amarin.org/professional/jumpstart.php>.
- [8] Burgess, Mark, "Recent Developments in Cfengine," *Unix.NL Conference*, Waardenburg, Netherlands, 2001.
- [9] Craig, Wesley D. and Patrick M. McNeal, "Radmind: The Integration of Filesystem Integrity Checking with Filesystem Management," *Large Installation System Administration Conference*, 2003.
- [10] Oetiker, Tobias, "TemplateTree II: The Post-Installation Setup Tool," *Proceedings of the 15th Systems Administration Conference (LISA)*, <http://isg.ee.ethz.ch/tools/tetre2/pub/tetre-lisa.pdf>, 2001.
- [11] Fink, <http://fink.sourceforge.net/>.
- [12] Oetiker, Tobias, "SEPP – Software Installation and Sharing System," *Proceedings of the 12th Systems Administration Conference (LISA)*, <http://people.ee.ethz.ch/~oetiker/sepp/>, 1998.
- [13] *ISC DHCP 3.0 Mac Netboot Patch*, <http://staff.harrisonburg.k12.va.us/~rlineweaver/macnb/>.
- [14] Bombich, Mike, *Mike's Mac OS X Management Software and Tips*, <http://www.bombich.com/>.
- [15] Apple Developer Connection, *Introduction to Bundle Programming Guide*, <http://developer.apple.com/documentation/CoreFoundation/Conceptual/CFBundles/index.html>.
- [16] *Introduction to Apple Events Programming Guide*, <http://developer.apple.com/documentation/AppleScript/Conceptual/AppleEvents/index.html>.
- [17] Schweikert, David, "ISGTC: an alternative to ~bofh/bin," *4th International System Administration and Network Engineering Conference*, <http://isg.ee.ethz.ch/publications/papers/isgtc-sane.pdf>, 2004.

