The *Depot*: A Framework for Sharing Software Installation Across Organizational and UNIX Platform Boundaries

Kenneth Manheimer – NIST
Barry A. Warsaw – Century Computing
Stephen N. Clark – NIST
Walter Rowe – NIST

ABSTRACT

The *depot* is a coherent framework for distributing and administering non-OS-distribution UNIX applications across extensibly numerous and diverse computer platforms. It is designed to promote reliable sharing of the expertise and disk resources necessary to maintain elaborate software packages. It facilitates software installation, release, and maintenance across multiple platforms and diverse host configurations.

We have implemented the *depot* using conventional UNIX subsystems and resources combined with policies for coordinating them. This paper presents the specific aims, structure, and rationale of the *depot* framework in sufficient detail to facilitate its implementation elsewhere.

Keywords: *Depot*, UNIX, sharing, distributed file system, `/usr/local`, installation, third-party.

Introduction

Installing and administering third-party UNIX applications often requires significant investment of time and expertise, precious commodities in any organization. Duplicating this investment is usually not the most efficient way to distribute its benefits. Instead, it's much preferable to share the product of this investment in the form of stable, usable configurations, provided organizational and platform discrepancies between different machines can be overcome. The *depot* is a systematic organization for distributing the products of expert application maintainers' efforts in an efficient and unburdensome manner. The foundation of this system is a generalized framework for installation and maintenance of applications that accommodates distribution across multiple platforms in a versatile way.

With the greater distribution that this framework provides, reliability and change-release management become more critical. The *depot* has comprehensive provisions to reduce and sometimes eliminate difficulties inherent in greater operational interdependencies between hosts.

Depot Objectives

The *depot* provides a mechanism for distributing application installations across numerous machines. In order to be successful, it must accomplish this while meeting the following criteria:

- Generality: Accommodate diverse UNIX operating systems, hardware platforms,[1] and host configurations as well as diverse application packagings. Commercial, academic, and public domain packages each come with their own often elaborate installation methods and mechanisms and we need to accommodate them all.
- Robustness: Provide predictable and consistent services. Formalize procedures for staged release of new packages and new package versions.
- Scalability: Provide for incremental addition and commissioning of applications, clients,

[1] To date, the *depot* has been implemented only on various Sun Workstation architectures, but no essential mechanisms are Sun-specific. Our implementation makes extensive use of "conveniences" like Sun's *NIS* distributed administrative databases and Sun's *automount*[2]. *NIS* is becoming universally available, and automount capability is widely available as *amd*[3] for many UNIX and some non-UNIX platforms.

and servers to the extent that the underlying distributed filesystem allows.
- Reliability: Use reliable distribution mechanisms and support redundant fallback copies.
- Ease of use: Be easy to commission and employ. Avoid burdening either application administrators or users due to *depot* involvement.

What the *Depot* Is Not Intended to Do

The *depot* is not a project management system. Although it provides for staging software updates and releases, it is not intended for, nor is it particularly suited to, multi-agent source modification. It is best used for distributing software installation and upgrades, and not for software development itself.

The *depot* is not intended to replace usual conventions for software sharing but instead refines and complements their functions. The `/usr/local` directory hierarchy is typically used as a repository for installing non-core utilities and incidentals. Often this hierarchy is shared across clusters of hosts that are similar both in operating platform and in general organizational configuration and use. With the addition of the *depot*, `/usr/local` can continue to be used for those items specific to a distinct homogeneous cluster of machines. Those items warranting broader service across organizational and/or platform boundaries (and additional intrinsic rigors of modification and release) belong instead in the *depot*.

Depot Motivations

There is useful software that is not included in UNIX OS distributions.

Typical UNIX-based software development efforts require programming and other special-purpose tools that aren't part of the core OS distribution. For instance, at our site we use and maintain our own copies of freely available software such as the Gnu Project tools[7] and the X Window System[5], as well as numerous homebrew tools developed locally or floating around Usenet. We also use various third-party and "unbundled" software utilities, like commercial databases and publishing toolkits, that perform functions which are either not available in core OS distributions or only provided for in a rudimentary fashion.

Integrating such tools incurs substantial costs in expertise and other resources.

Expertise is expensive and must be applied efficiently. Non-OS software products, not delivered with OS distributions, often demand specialized expertise to maintain and accommodate them. Even well produced and packaged commercial products require disk space and expertise for their management. These products must be integrated with, and maintained in the context of, existing installations, which may already be specially tailored with diverse customizations.

Diversity can be an obstacle to sharing.

Large workstation-based computing sites generally consist of similarly configured subclusters of affiliated workstations. It is relatively straightforward to arrange to share distribution OS and other applications among the similarly configured members of one of these subclusters. (For example, it's quite common to find similar machines sharing network mounts or duplicates of a `/usr/local` filesystem that houses non-core applications.) However, differences between the configurations of machines in different clusters, or differences in OS revision, vendor, or hardware platform between machines that are otherwise similarly configured, thwart such direct approaches to sharing.

In particular, applications that can be prepared for diverse platforms usually require certain relationships between their executables, libraries, and other incidentals to be preserved across hosts. For instance, Gnu Emacs needs to know where to find its runtime lisp libraries, ancillary executables, and online documentation. More generally, many applications include runtime dynamically loaded libraries that need to be located in specific places for the applications to find them. Ad hoc sharing schemes developed for specific differences between specific machines will often fail to extend to other differences on other machines.

Generalized schemes may provide wider service at the expense of greater restrictions on client configurations. The challenge is to exploit sharing capabilities without imposing undue complexity or interference either on the existing individual host operating environments or on the experts administering the applications. In general, we don't want the savings in duplicated expertise required to manage the distributed software to be defeated by costs of accommodating or managing the distribution methods themselves. A good arrangement can avoid these pitfalls without compromising the benefits.

What the *Depot* Really Does

The *depot* is a network-filesystem based organization for sharing application installations across UNIX-based platforms. Most importantly, applications are easy to install and use from the depot. "Depotized" applications are arranged to be self-contained and structurally consistent across platforms so that internal relationships among application components are preserved regardless of the organization or platform of the client hosts. The *depot* avoids introducing undue dependencies between applications and their surrounding operating environments, or vice versa, and it does not interfere with intrinsic dependencies already present in an application.

Design Overview

Two abstract objectives have crucial influence over the shape of the *depot*:

- Provide transparent accommodation of multiple platforms
- Maximize generality; minimize dependence of depotized applications on the surrounding operating environment and on each other

Transparent accommodation of multiple platforms is accomplished by mapping from pan-platform server arrangements to platform-specific client arrangements.

While some third-party application packages accommodate multiple platforms with a single installation, most do not. Multi-platform installation and employment could be taken care of separately with platform-independent scripts. However, such scripts do not normally generalize from application to application. Indeed, it's difficult to arrange for the same script to take care of both installation and employment of even a single application. *Depot* structural arrangements instead transform an internal pan-platform arrangement of an application on servers to a public platform-specific arrangement on clients.

Separate directories are allocated in the pan-platform arrangement for the platform-specific portions of an application. Clients mount the entire pan-platform arrangement and then overmount the correct platform-specific components in a slot set aside for that purpose. Since the platform-specific components are effectively organized in the same way for all platforms and the platform-independent components are shared between all platforms, each client sees the same structural organization regardless of its platform. Only the platform-specific files themselves are different. This arrangement, as far as the client is concerned, looks like a configuration suited for installation and employment of the application on the client's own platform.

Arranging for simple mount schemes and minimizing dependence between *depot* applications and their operating environments dictates strong emphasis on self-containment.

Interdependencies between an application and its operating environment complicate the job of making the application widely available across diverse environments. In order to minimize this complexity we keep the arrangement of an application installation very self-contained. This self-containment is essential to avoid imposing unnecessary burdens on clients or application maintainers who use the *depot*. Dependencies of an application's installation on the structure of a client's operating environment are kept to a minimum, and, conversely, the *depot* design strenuously avoids imposing restrictions on the client's operating environment.[2]

Each depotized application is contained within a single directory hierarchy. The contents may be composed from mounts of scattered filesystems,[3] but they collectively look like a single hierarchy. The collection of *depot* applications is likewise contained within a single directory hierarchy. Those components that an application intrinsically requires to be established elsewhere in the operating environment are represented in the external locations by symbolic-link proxies that point to the actual components in their locations within the *depot* hierarchy.[4] (These links should be created by a script prepared as part of the process of incorporating an application into the *depot*, to ease commissioning of new clients.)

Implementation

The root of the *depot* hierarchy is located in the same place on clients and servers. All of the paths configured into depotized applications are prefaced by the path of the hierarchy's root, so a short one is preferable. We have our *depot* root located at `/depot`.

A *depot* installation of an application has two principal aspects: the arrangement of disk storage for the pan-platform components of the application, and the public interface to it. The platform-specific public interface is implemented on every client that subscribes to the *depot*, and we will refer to it as the "client view". The client view is composed, using *NFS* mounts, loopback mounts, or symbolic links, from the pan-platform arrangement, which we will refer to as the "origin view".

A host that serves as an origin for an application (or for a piece of it) usually also makes use of the application as a client and so employs both origin and client views. (The converse, however, is not true: the majority of clients do not serve as application origins.) Together with the mechanisms that we use to compose the client view from the origin view, this requires the *depot* location of the client view to be different from the *depot* location of the origin view.

---

[2]Note that *depot* packagings for an application *may* include shortcuts that involve nonessential client dependencies, just so long as their functionality is available in other, non-constraining ways.

[3]For instance, sometimes filesystem service of an application's platform-specific components is distributed among different hosts, with each host serving only the components which are specific to its own platform.

[4]X11 under SunOS 4 contains an example of an application with components that need to be located outside of the *depot* hierarchy. *Xterm* depends on a dynamic library which must be located in either `/usr/lib` or `/usr/local/lib` for setuid-authorization purposes.

We will first detail the arrangement of the origin view. Next, we will describe the public interface provided by the client view, and finally, the mapping from the origin view which is used to implement the client view.

The Origin View

The origin view contains all of an application's components, including a single copy of each of the platform-independent components and a copy of each of the platform-dependent components for each of the supported platforms. It is not intended, however, to be directly usable for either installation or execution of the application - this is the role of the client view.

Within the *depot* root, application origins are located in subdirectories whose pathnames begin with either `/depot/.primary` or `/depot/.develop`.[5] These two directories differ only in the way they are used; their internal organizations are identical. Fully released, in-service application copies are situated in `.primary` directories. The `.develop` directories provide private, temporary work areas in which to perform *depot* application builds or experiment with changes without affecting other users. (See "Isolating Release Preparations for Upgrades" below for more details.)

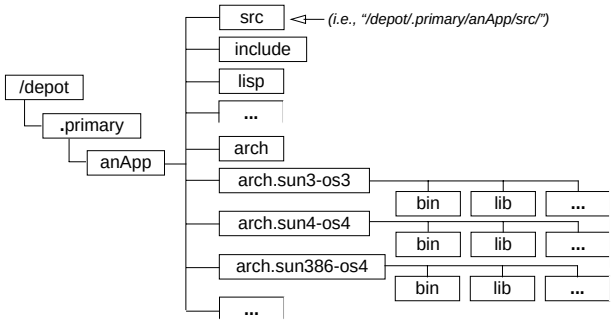


Figure 1 – Origin View of *anApp*

Figure 1 shows a portion of the origin arrangement of a fictional application named *anApp*. Each box represents a directory (or collection of directories, in the case of the boxes containing ellipses). Sibling directories on the path above `/depot/.primary/anApp` are ignored.

- The `src` directory contains the source distribution for *anApp*.
- The `include` and `lisp` directories are typical examples of subdirectories containing platform-independent components.
- The `arch` directory is a stub necessary for use in constructing the client view.
- `arch.sun3-os3`, `arch.sun4-os4`, and `arch.sun386-os4` are typical examples of directories which contain platform-specific components. They commonly have subdirectories `bin` (for public executables) and `lib` (for public and internal object libraries).

It is common to have separate `lib` directories for platform-independent and for platform-dependent components. The platform-independent `lib` might contain ASCII text files like default and "rc" configuration files, skeleton files for code generators, etc., while the platform-dependent `lib` would hold object code libraries and byte-order-sensitive files like fonts.

The name of each platform-specific directory distinguishes the platform to which it belongs. It is only necessary to distinguish between fundamental OS, hardware executable format, or byte-order incompatibilities.

Each platform-specific directory name starts with the prefix "arch."[6] The next few letters indicate the hardware architecture of the supported platform. Finally, the string "-os" is concatenated with a string that indicates the supported operating system. Thus, for example, `arch.sun4-os4` denotes the directory for software specific to Sun SPARC ("Sun4") architectures running SunOS 4.

Depot servers need to grant at least remote read-access privileges for clients to mount the origin directories. Since compilation and installation are done within the client view, clients used for *depot* administration must have read/write privileges. We use "read-mostly"[7] together with root-access designations to grant suitable privileges to the specific clients that will be used for building while restricting all other clients to read-only. This assures that only authorized clients can be used to make changes to the applications.

The Abstract Client View

A client view of an application is a platform-specific arrangement employed for both administration and public use of the application on a particular machine. Composed from the pan-platform origin view using *NFS* mounts, loopback mounts, or symbolic links, in the abstract the client view looks like a dedicated installation of the application for its host platform.

All path references for the application, whether for internal configuration or for public access, use paths dictated by this abstract arrangement. Thus it provides both the public interface to the application and the internal interface between its components.

[5] The dot '.' prefixes are not so much for the ostensible (and rather thin) UNIX purpose of "hiding" these directories, but rather for the sake of distinguishing them from the other contents of the directory by clustering them together at the front of *ls* listings.

[6] "arch." is a holdover from early *depot* days; "plat." or "platform." probably would have been more appropriate.

[7] SunOS *exportfs* (8) man page[1].

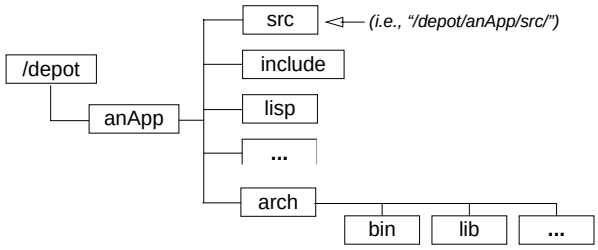


Figure 2 – Abstract Client View of *anApp*

The Origin-Client Mapping

The mapping between the origin view and the client view is the crux of the *depot* scheme. The client view is composed by mounting the server arrangement and then overmounting the suitable architecture onto the empty `arch` directory. In the absence of *automount* and loopback mounts origin servers can use symbolic links to achieve this client mapping locally. (The implementation of the mapping is explained in detail in "Implementing the Client View", below.)

Using our example, the *anApp* root origin would be `/depot/.primary/anApp` on some host. This origin is mapped to `/depot/anApp` on the client. Next, the particular `/depot/.primary/anApp/arch.<arch>-<os>` directory suited to the client platform is mapped to the `/depot/anApp/arch` stub directory, provided specifically for this purpose. As a result of this mapping the `arch` directory on the client effectively contains the platform-specific components of the application required by the client's platform.

The resulting arrangement on the client is illustrated in Figure 3. It shows the typical arrangement of a *depot* client, including the location of the origin root directory on those clients that also serve as *anApp* origins. Each directory is represented by a box whose shading indicates the role it plays in the arrangement and in the mapping.

- **Plain Local** (`/depot`) are regular directories in the root of the local file system.
- **Local, only on Origin servers** (`.primary`, `.develop`) are directories that are present on clients only if they happen to be origin servers. ("The Origin View" section, above, details their contents.)
- **First Redirects** (`anApp` and its subdirectories) are established by a mount of or link to the root directory of the *anApp* application on the origin server (`/depot/.primary/anApp`).
- **Overlaid Redirects** (`arch` and its subdirectories) are established with a second mount from the `anApp` origin hierarchy onto the empty `arch` directory. The mount maps the particular platform-specific directory for the host (in this example, `arch.sun4-os4`) into the `arch` directory of the client view.
- **Other Redirect Stubs** (`X11`, `yAp`, `zAp`, `...`) are shown simply to illustrate that clients may subscribe to numerous applications. Structural details are not shown but would follow the same principles illustrated by `anApp`.

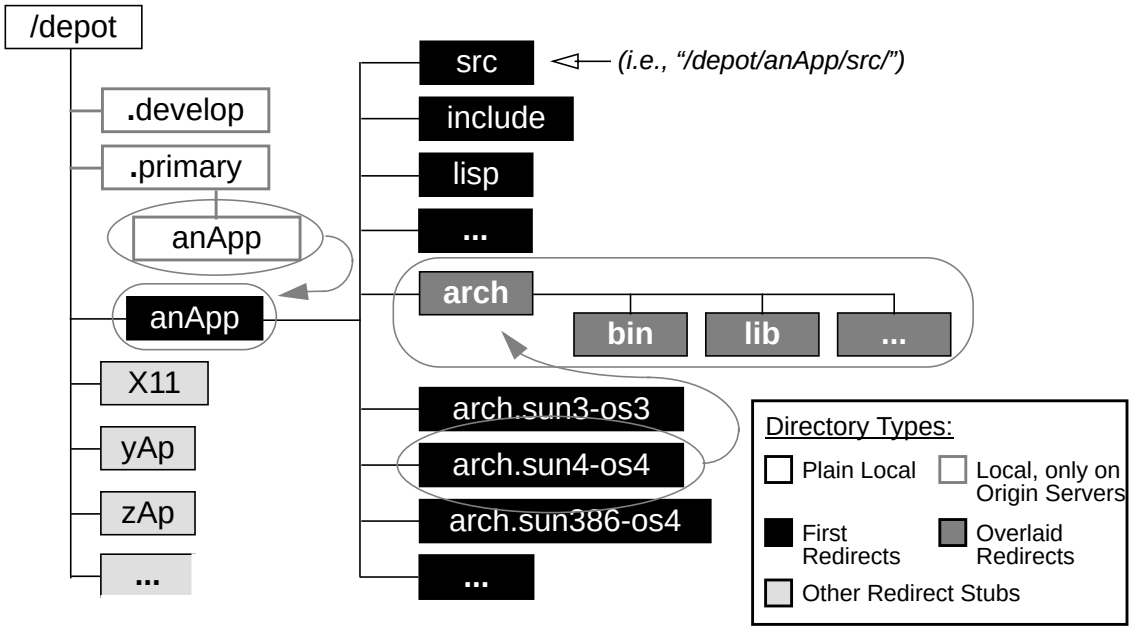


Figure 3 – Origin/Client Mapping of *anApp* on a Sun4-os4 Host

As mentioned above, the only important `arch*` directory in the client view is `arch.` This directory effectively contains the platform-specific components of the application, and the `arch.<arch>-<os>` directories are ignored.

Both application configuration and public references to application components should resolve to the `arch` directory. Public access to any platform-specific components should either refer directly to `/depot/anApp/arch` subdirectories or get to them via symbolic link proxies. In this way diverse clients use what appears to be the same structure to resolve both platform-dependent and independent application components, and both installation and employment of an application use the same paths regardless of the client's platform.

Implementing the Client View

Sun's *automount* significantly simplifies implementation of the origin/client mapping.[8] It provides the means to systematize and distribute mount configurations via a networked administrative database (*NIS*). It also accounts for special requirements of hosts that serve as both origins and clients of an application. In Appendix I we include a representative *automount* map to help illustrate how to use *automount* for *depot* purposes. Below we detail the non-*automount* procedure for implementing our *anApp* example, both for the sake of clarifying the origin/client mapping and to show how to implement it when *automount* won't be used.

We'll use the syntax of Sun utilities for our example. The Sun4 host `honcho` running SunOS 4.1 will serve the application *anApp* from the origin directory `honcho:/depot/.primary/anApp`. (*anApp*'s directory structure on `honcho` would look much like the skeleton exhibited in Figure 1.) The Sun3 client `guppy`, running SunOS 4.0.3, would compose its client view of *anApp* with the following two lines in `guppy:/etc/fstab`:[9]

```
honcho:/depot/.primary/anApp \
        /depot/anApp      nfs rw 0 0

honcho:/depot/.primary/anApp/arch.sun3-os4 \
        /depot/anApp/arch nfs rw 0 0
```

It is usually necessary to implement client views on origin servers, and some implementations of NFS are prone to serious failures when mounting from a server to itself. Loopback mounts under SunOS 4 are one solution to the problem. If they are available, loopback mounts can be used to establish the client view on the origin server by using `fstab` lines like those above but replacing each occurrence of "nfs" with "lo".[10] Symbolic links can be used instead to implement this arrangement in another way. Here are the appropriate link commands for this method:

```
ln -s /depot/.primary/anApp \
        /depot/anApp
ln -s /depot/.primary/anApp/arch.sun3-os4 \
        /depot/anApp/arch
```

The first link simply creates a redirection from the client view `/depot/anApp` directory to the application origin at `/depot/.primary/anApp`. The second link creates a redirection from the client view `/depot/anApp/arch` directory (which is identically the origin view directory `/depot/.primary/anApp/arch`, as a result of the previous link) to the platform-specific origin directory `/depot/.primary/anApp/arch.sun4-os4`. Note that if the platform-specific stub, `/depot/.primary/anApp/arch`, already exists, the second link will not be properly created, but will instead be placed inside this stub directory.

There is a somewhat obscure complication in the link scheme which turns out not to be a problem but which bears explaining nonetheless. Since the second link is actually established in the origin view as the `arch` redirection, it's seen by all clients that mount this filesystem. The question is how this affects a client that uses mount to redirect the arch directory (in this case the link) to the appropriate platform-specific directory.

Since the link is resolved at mount time, the desired `arch.<arch>-<os>` directory is mounted over whichever platform-specific directory this link points to. The `arch` link is therefore not covered by the mount, and points to the overmounted `arch.<arch>-<os>` directory rather than the shadowed (covered) one. Therefore references within the `arch` hierarchy resolve to the desired `arch.<arch>-<os>` platform-specific components.

Automount-based implementations will also cooperate correctly with origin/client links, so all three methods can be used in parallel without conflict.

Using Origin Redundancy to Increase Reliability of *Depot* Services

Application origin directories can reside on any fileservers and they can be divided between combinations of fileservers. By establishing alternative copies of entire origin hierarchies we provide the basis for both staged release of software upgrades

[8] Although our implementation uses *automount*, *amd* also seems to be at least sufficient for our purposes.

[9] Note that Sun's *mount* does dependency analysis before processing mounts, so these hierarchical mounts pose no problem.

[10] Loopback mounts must be last in `/etc/fstab`; see the warnings section of the SunOS 4 *mount*(8) man page[1].

and fallback redundancy to increase reliability by reducing critical points of failure.

Multiple `.primary` Origins Provide Redundancy

By creating multiple `.primary` origin hierarchies for crucial applications we are able to achieve distributed loading of the origin servers and, perhaps more importantly, provide fallback service in the event of a fileserver failure.

Since there are multiple servers for complete origin copies, a client will have alternates from which it can get the software if an origin server goes down. At worst a client will need to reboot to free itself from a locked-in mount. (A mount can become locked in when an active executable can't be completely terminated because it's hanging on a disk read from the defunct filesystem.) Once the client is freed it can redirect its mounts to surviving alternate servers. With *automount* the rebinding process is automatic, though locked-in mounts may prevent rebinding, so that reboot may sometimes still be required.

Isolating Release Preparations for Upgrades

For tools that are perpetually in use, like Emacs and X, it's important to minimize down time. The building and testing phases inherent in controlled releases of new versions can be time consuming. By arranging for a client to use its own copy of an application origin the release can be thoroughly prepared in isolation.

Using one of multiple `.primary` origins is not suitable for this purpose. Publicly enlisted alternate `.primary`'s must be kept synchronized and it is awkward (and usually undesirable) to quickly remove a publicly enlisted origin from service. Instead we establish distinguished `/depot/.develop`[11] copies specifically for preparing the release, doing the build and testing in isolation from the rest of the world. Once the release is prepared it is announced and migrated as a whole to the `.primary`'s. (The `.develop` versions can then be deleted, although they may be useful as placeholders to reserve disk space for the next release cycle.)

Some Incidentals on Implementing Redundancy

We use a special *automount* map akin to the Sun "`-hosts`" map that allows us to see the entire origin structure of all servers to facilitate copying the contents of the `.develop` origins to the corresponding `.primary` origins. This is especially useful when we use multiple `.primary` origins for an application to provide redundancy.

[11] Another holdover from initial development which unfortunately implies something other than what we mean. This might more appropriately be called something like `.aside` or `.scratch`.

We have resolved some formal policies about management of the `.primary` and `.develop` application origins in order to facilitate cross-divisional use of the *depot*. Most importantly, multiple `.primary` origins for an application are guaranteed to be held as consistently identical as can be managed. This is necessary to ensure that clients can rely on identical service from any of the `.primary` copies of the application. It is crucial when using *automount* with multiple primaries because *automount* does not necessarily use the same host for **first redirects** and **overlay redirects**, and will combine platform-independent and platform-specific components of an application from separate servers.

Also, it's important to identify managers for each application who will at least coordinate additions and upgrades to it. We stipulate that any changes of applications must be arranged with the designated application administrator, and any potentially disruptive releases to `.primary` origins should be done with the direct involvement of the administrator. Furthermore, as with any system changes that impact users, any potentially disruptive changes should be scheduled to the satisfaction of the range of clients using the applications.

Results

We have been developing the *depot* for about a year now and have been using it in near final form for the last half year. Its use spans two major organizational divisions of our laboratory, and will soon include a third. We use it to serve numerous applications to a contingent of more than one hundred workstations, at one point including seven distinct operating "clusters", nine comprehensive file servers, and three major OS versions.

Most dramatically, both divisions have a larger repertoire of better maintained utilities thanks to their availability through the *depot*. We use major research and academic programming tools including X, NeWS, Gnu, InterViews, and Usenet news facilities, and numerous commercial products including FrameMaker, Saber-C, Parasolid, Hoops, and Allegro Common Lisp. We have consistently maintained an up-to-date repertoire of all of these tools across two major OS releases (Sun OS 3.5 and 4.0/4.1) and three different hardware architectures (Sun-3/68020, Sun-386i, and Sun-4/SPARC) with only a single central administrator for each application (two for Gnu - one for Emacs and one for the rest) providing service for both divisions.

Redundant origin hierarchies are big wins. By dedicating some disk space to additional origin hierarchies, reliability can clearly be enhanced well beyond what would be available with a single origin. In the case where multiple clusters are sharing services, redundant hierarchies may not even require extra disk space - it is likely that each cluster already maintains its own copy of an application, so

that all that is required is to implement *depot* disciplines on the various hosts.

The consistency of depotized application distribution makes one copy interchangeable with another, while separately managed versions usually are not trivially interchangeable for the reasons cited above (see "Diversity can be an obstacle to sharing"). Large groups of clients can be served by relatively few copies, so the returns improve up to some fairly high server or/and network loading (or even connectivity) saturation point as scale increases.

By establishing duplicate primaries for important *depot* applications on mutually independent cluster servers we've achieved much better uptime. In particular, because of the immediate interchangeability of the duplicated applications, we can have one server off-line (either intentionally or due to a system failure) and only those machines dependent on boot services or on applications not incorporated into the *depot* are incapacitated. During four separate major system failures over the past year we reduced what would have been down-time for some major applications (X, Emacs, FrameMaker) for at least thirty machines (and up to sixty machines, depending on which division's machine was hit) to down-time for only a maximum of ten dependent boot-clients. Considering that the repairs on one of those occasions stretched out to over a week, that constitutes a major reduction in lost work-hours.

Perhaps the most outstanding sign of the success of the *depot* is the degree to which the respective divisions' managements allow this cross-dedication of talent to each other's facilities. We feel that the only reason we are able to "get away" with this is because they recognize, as do we, that we're all getting more comprehensive and thorough service with less invested effort and greater ease of use than we did prior to the commissioning of the *depot*. And that was while we were still developing it . . .

Summary

The *depot* provides a framework for installing arbitrary software, including third-party and custom applications, to accommodate diverse platforms. The structural arrangement of an application's installation is consistent from one platform to another, allowing the same usage and installation path across platforms. Applications pre-packaged with multi-platform accommodations are simply installed without fuss. Commissioning an application's *depot* installation requires no more finagling than does adapting multiple copies of the same configuration for installation on multiple standalone machines, and usually requires less effort if the standalone machines don't happen to be identically arranged.

All components of the *depot* except for the inherently public application components (the user interface) are confined to a single directory hierarchy on any client's file system. Simple relationships among the actual installed components hold regardless of the host file system environment. Even the external interface is implemented as a simple, reproducible, and platform-independent entity. Thus commissioning an application in the *depot* usually entails establishing a small set of filesystem mounts, establishing the application-mandated hooks if any, optionally establishing symbolic link surrogates for the external interface for access, and creating a script to automatically create all of the necessary external links identified in this process (this is to ease the commissioning of new clients). This almost always winds up being even simpler and cleaner than it sounds.

The consistent organization of *depot* sharing allows redundancy to be used directly to increase reliability. As scale increases the returns increase, up to the saturation point of the media (fileserver hosts, *NFS*, and/or network).

It is important to note potential problems that the *depot* avoids. It depends only on conventional UNIX utilities and imposes minimal overhead on the application servers, maintainers, users, and client systems, providing a robust basis for multi-platform support of diverse utilities. It does not interpose clumsy interfaces for installing or accessing platform-specific components of an application, relying instead on remounts which are almost entirely transparent to both application management and the user clientele.

Unresolved Issues and Other Work

- Applications with installed components that are not strictly partitioned from their source distribution require extra finagling for installation in the *depot*. For instance, X11r3's *imake*[6] mechanism required some extra effort in order to establish this partitioning, though X11r4 has solved that problem with the introduction of *xmkmf*. Gnu Emacs also exhibits the problem. It uses the distribution `etc` directory for ancillary executable components that are necessary both for build and operation of the application. It is necessary to build some custom scripts in order to implement the partitioning for Gnu Emacs. We think it may be reasonable to consider this partition between source distributions and built releases as one criteria of a "good" installation mechanism, but have to evaluate this further.
- We need to implement the *depot* on other non-Sun machines. While we have small numbers of various other UNIX platforms around, including Silicon Graphics, DEC, and IBM, none of the active *depot* development personnel are responsible for those machines. Now that we have reached a fairly stable

framework we intend to branch out a bit.

- We need to investigate newly available technologies, e.g. "translucent" file systems[4], and evaluate how we can use them to improve on the simplicity and transparency of the system.

Acknowledgments & Disclaimer

This work was jointly funded by the NIST Automated Manufacturing Research Facility (AMRF, project 734-3385) and Scientific and Technical Research Services (STRS, project 734-3106).

The *depot* scheme was initially conceived and designed by Ken Manheimer. Barry Warsaw and Ken refined the initial design. Barry implemented a prototype layout and Ken implemented the initial use of the overmounting scheme. Barry and Steve Clark developed specific methods for managing the Gnu software package as a whole. Walter Rowe did some similar work for sundry X tools. The initial layout, along with the conception of the *depot* in general, was further refined and resolved by the concerted efforts of all of the authors.

We are indebted to our collective management and to our numerous users in the Factory Automation Systems Division and the Robot Systems Division at NIST, who on numerous occasions had to put up with the growing and shaking-out pains of the progressively developing system. In particular, thanks to Scott Paisley, another local system manager, for valuable input and assistance, and to local guru Don Libes, who provided important criticism and insight while we were developing the depot and who encouraged us to submit and write this paper. (He was also the only person who had the guts to read early drafts of this paper.)

Appendix - A Representative Automount Map

The automount fragment depicted in Figure 4 illustrates some nuances of Sun's *automount*, particularly the combination of hierarchical mounts and alternative servers for a common hierarchy.

Note that hierarchical automounts composed from alternative servers can be and often are realized with components from both servers. For example, it is not unusual to find the `/depot/gnu` directory mounted according to the above fragment to come from the host `imp` and the `/depot/gnu/arch` directory to be mounted from `dip`. For this and other reasons it is imperative that the alternative origins be held in strict synchronization.

References

[1] Sun Microsystems Incorporated, *SunOS 4.1 Reference Manual*.

[2] Sun Microsystems *SunOS 4.1 System Administration Guide* or *SunOS 4.0.3 System Administration Addenda* is an essential supplement to the *man* pages.

[3] Jan-Simon Pendry, "Amd - An Automounter", Department of Computing, Imperial College, London, England, 1989.

[4] Sun Microsystems Incorporated, "TFS", *SunOS 4.1 Reference Manual*, Vol 2, p. 1494.

[5] Scheiffler, R.W. and J. Gettys, "The X Window System", *ACM Transactions on Graphics* Vol. 5, No. 2, April 1986, pp. 79-109.

[6] Jim Fulton, "Configuration Management in the X Window System", The MIT X Consortium, MIT, Cambridge, MA, 1989.

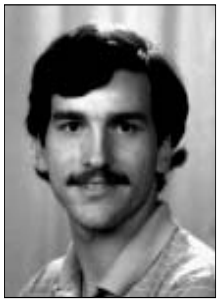
```
## Note: We *cannot* include entries that cause a
#  dir to go on top of itself.
#target root    dir/opts    <Sys>:<path> origin
#-----------    --------    ------------------
/depot/autotabs / -ro       elf:/depot/.primary/autotabs
/depot/sundry   /           elf:/depot/.primary/sundry \
                /arch       elf:/depot/.primary/sundry/arch.sun3-os4
/depot/X        /           imp:/depot/.primary/X \
                            dip:/depot/.primary/X \
                /arch       imp:/depot/.primary/X/arch.sun3-os4 \
                            dip:/depot/.primary/X/arch.sun3-os4 \
                /src        dip:/depot/.develop/X/src
/depot/gnu      /           dip:/depot/.primary/gnu \
                            imp:/depot/.primary/gnu \
                /arch       dip:/depot/.primary/gnu/arch.sun3-os4 \
                            imp:/depot/.primary/gnu/arch.sun3-os4
```

Figure 4 – Automount Fragment

[7] Available from The Free Software Foundation of Cambridge, Massachusetts, further information is available via electronic mail on the Internet from gnu@prep.ai.mit.edu.

Ken Manheimer works as UNIX Systems Support Manager in the Factory Automation Systems division at National Institute of Standards and Technology, where he has shepherded the growth of his divisions UNIX computing from four Sun 1's (and Eunice on a VAX) to seventy+ UNIX systems. He received a B.A. in Computer Science from Hampshire College. Reach him at NIST; Bldg 220, Rm A127; Gaithersburg, MD 20899 or electronically at klm@cme.nist.gov.

Barry A. Warsaw has just recently joined Century Computing, Inc. as a Data Systems Engineer, where he will be working on an online retrieval system for the National Library of Medicine. Formerly with NIST, he was at times system manager for the Robot Systems Division network of UNIX machines, and developer of user interfaces for robotic and automated machine control systems. He received a B.S. in Computer Science from the University of Maryland. Reach him at Century Computing; 1014 West Street; Laurel, MD 20707 or electronically at baw@fox.gsfc.nasa.gov.



Stephen N. Clark has never been a system administrator in his life, falling instead into the amorphous category of "knowledgeable user." He is currently working on tools for building schema-driven applications in support of the National PDES Testbed at NIST. He received an Sc.B. in Math and Computer Science from Brown University. Reach him at NIST; Bldg 220, Rm A127; Gaithersburg, MD 20899 or electronically at clark@cme.nist.gov.

Walter Rowe is currently the System Administrator for the Robot Systems Division of the NIST, where he maintains a network of 30 Sun workstations. He received a BS in Computer Science from Tennessee Technological University and is currently working on a MS in Computer Science at the Johns Hopkins University in Gaithersburg, Maryland. Reach him at NIST; Bldg 220, Rm B124; Gaithersburg, MD 20899 or electronically at rowe@cme.nist.gov.