Towards a POSIX Standard for Software Administration

Barrie Archie
ICL

Towards a POSIX Standard for Software Administration

Barrie Archer – ICL

ABSTRACT

The POSIX draft standard for Software Administration is about to go to ballot for acceptance as a formal POSIX standard. Since this standard is likely to form the basis of future Software Administration products it will have a profound effect on the facilities available to administrators and the way they manage software. This paper explains how the standard came about, gives a summary of the features and explains how systems administrations can, via the balloting process, have an influence on the final standard.

Introduction

The distribution, installation and control of software is an important and time consuming task for administrators. Most vendors supply tools for their own systems, but, especially in a network of heterogeneous systems, administrators have often had to resort to inventing their own methods. Previous papers at LISA have reported on some of these efforts. To address this problem the POSIX Systems Administration Group (P1003.7) set up a subgroup to propose a standard for software administration. Working from the specifications of existing tools this subgroup produced a draft standard that will shortly be balloted for acceptance as a POSIX standard.

The purpose of this paper is to bring to the notice of a wide audience the impending ballot of the draft standard and to encourage participation in the ballot as well as to explain what is in the draft standard and why. In describing the draft standard more emphasis is put on the overall structure and the background to what is there, than expounding the detail. By doing this it is hoped that reviewers will appreciate the conflicts that were addressed and the process that led to their resolution. They will then be in a better position to understand what the draft standard is trying to achieve and will be able to contribute to maintaining a coherent standard.

This paper was prepared whilst the draft standard was still under development and so anything stated here should be taken as a guide only - refer to the standard itself for definitive information. Also, for the sake of readability, some simpler terms have been used in this description in place of the formally defined terms in the draft standard.

Objectives

The subgroup defined three objectives that the standard should address.

Administrator Portability

By providing an interface for software administration that was consistent on all conformant implementations, administrators would be able to use any such system without retraining.

Standard Packaging format

A common packaging format would enable software to be processed on any conformant system. This does *not* imply a architecture independent format, although it does not preclude it. Software can only be run on an architecture it is designed for. It does allow, however, for discless clients to be catered for.

Distributed Administration

The provision of interoperability interfaces enables distributed software administration across systems. This can be done either through the command line interface or through a management application specifically written for the purpose.

Standards

In order to be useful, a standard must define interfaces or formats in a sufficiently rigorous way that there should be no ambiguities that could result in incompatibilities between implementations. How this applies to POSIX standards is discussed in a later section. However, attaining this necessary rigour does not lead to a readable document. For example, any particular aspect should only be defined once in a standard, whereas for readability a summary of the aspect might appear in several places.

Rationale

In order to try to address the problem of readability POSIX standards contain sections of *rationale*. These sections are intended to explain what parts of the standard mean, how they are expected to be used and why they are there. Even the addition of rationale has its limitations, however, and cannot substitute for the kind of overview being presented here.

Scope

Another important consideration in defining a standard is to limit its scope to something that can be achieved in a reasonable time. There is a trade off here between what one would like to do and the least one can do for a usable standard. In the section on the history of this standard there are some comments on how the scope changed over the definition life cycle. An aim of this paper is to give some information about how the standard came about and why it covers some things but not others. It is hoped that this will enable those who join the balloting group to be in a better position to make comments.

POSIX Standards

POSIX Standards have to be approved by the Project Monitoring Committee who will seek to assure that the standard is reasonable, that it is based on existing practice and that there is sufficient support to enable the work to be done. Once such approval is obtained a group (or sub-group as in the case of Software Administration) can be formed which will meet at the quarterly POSIX meetings to progress the development of the draft standard. At the end of the development process a draft will be produced which will be balloted

Balloting

To ballot a draft standard a balloting group is formed. The IEEE uses appropriate means to advertise that the group is being formed. Any individual may join, but comments from those who are not members of IEEE or the Computer Society are for information only. To pass the ballot 75% of the balloting group must respond and 75% of those responding must agree to the standard. Agreement can be the result of comments being taken into account - the process known as ballot resolution. Of, course if there are too many comments requiring material changes it would be necessary to ballot again.

Mock Ballot

It is customary for groups to engage in a *mock ballot* prior to the ballot described above. The intention is to address the same audience and to find out if there are any fundamental problems before going to ballot. For Software Administration the mock ballot showed that Configuration, Recovery and Software Service (patching) would have to be addressed in the draft to go to ballot.

ISO Standards

Once standards have been approved through the balloting process they go forward to ISO for ratification as international standards. This is handled by Working Group 15 of SC22. There are certain agreements in place between IEEE and ISO designed to smooth this process by ensuring that the POSIX standards will be acceptable to ISO without alteration. One area where this affects the work is that a POSIX standard can only reference other formal standards. It cannot reference or rely on an implementation or de facto standard.

History

This section covers the way in which the draft standard evolved. This is useful information for understanding why the draft standard contains what it does and why the facilities are defined in the way they are.

Participation

One of the conditions of starting out on the process of producing a POSIX standard is that there should be sufficient commitment to enable the work to be done. The subgroup was fortunate that there were was a high level of commitment by several companies and individuals. Most major vendors were represented as were users, in the form of living/breathing systems administrators. The subgroup was also able to get work done between meeting by the use of a mail reflector. In this way even those who were not able to attend a particular meeting could continue to contribute.

Existing Practice

Another condition for a POSIX standard is that it should be based on existing practice and not be a invention of the group, hence indicating that the standard can be implemented. One of the first actions taken by the subgroup, therefore, was to examine the existing practice, and this was done by inviting submissions, either or both of a paper submission or a presentation. The companies that made such submissions are given in Table 1.

IBM
ICL
Digital Equipment Corporation
Hewlett Packard
SNI
SCO
UNIX Systems Laboratories

Table 1:: Companies making submissions

The subgroup found that all the submissions had many features in common and that there was a good deal of agreement in the facilities that should be provided. Obviously, some features were only found in some submissions and there were some misalignments. Nevertheless, the subgroup was encouraged by this to proceed, in the belief that there was a good chance that the interested parties could come to an agreement on a draft standard.

It should be noted that the requirement for existing practice brings its own complications. These can arise because existing practices in different areas being addressed by the draft standard do not fit together well, or because there is no one existing practice that gives all the facilities identified as necessary for the draft standard.

Comparison

The subgroup drew up comparisons of the documents submitted in order to determine the core facilities and to examine the additional aspects of particular submissions. Part of this work appeared in the rationale of the draft that was basis of the Mock Ballot.

Base Document

In order to start work on the text of the draft standard, the subgroup decided to adopt one of the submitted papers as a base document. The one chosen was the SDU utilities submitted by Hewlett Packard (also the basis of OSF DME software distribution). Having adopted this base document the group then proceeded to modify it so that it was more generic and also covered important features not found in the SDU utilities but which existed in other submissions or identified as needed by the mock ballot.

Mock Ballot

The document that was distributed for the Mock Ballot was Draft 8. It was recognised that a lot of work needed to be done on it before it could become a formal standard but it was felt that the time was appropriate to get a wider opinion of the work so far. 67 people took part in the Mock Ballot, sending in almost 1000 comments. Three of these responses were classed as votes against the proposed draft standard, the rest being qualified approvals. Many respondents identified the lack of rigour, but there were also many comments that pointed out problems that might not otherwise have been corrected before the formal ballot. In addition it became very clear that Configuration, Recovery and Software Service (patching) would have to be addressed to make the draft standard acceptable. In getting to draft 8 these items had been considered and dropped due to a lack of existing practice and in an attempt to simplify the task of producing the draft standard. What the Mock Ballot clearly showed was that many people saw them as vital parts of the standard.

Overview

This section gives a high level description of the draft standard, the details of which are filled out in later sections.

Components

The draft standard can be considered as consisting of three key components, which are required to achieve the objectives.

Packaging Layout

The draft standard defines the information that is held about software on a distribution medium, as well as the way this information is represented on the medium. This definition enables the use of different media to distribute software (including electronic transfer), optimising the use of each type of media according to its particular attributes. The draft standard does *not* define an architectural neutral format but does not preclude it. However, it does allow for the architecture of a product to be identified and for variants of the product for several architectures to be present simultaneously. Hence, an appropriate variant may be chosen from several on a medium.

Commands

The draft standard defines commands for performing the various tasks that are needed in order to perform software administration. The definition of these commands is based on the submissions received. On any conformant system an administrator will hence have a consistent way way of dealing with software.

Management Information

The draft standard defines the information which describes the software being managed. The draft standard does not define how this information is stored for software that has been installed, although it does define the way in which the tasks use the information. The management information is sometimes referred to as the Management Information Base (MIB) by analogy with Network Management standards, although this term will not appear in the draft standard.

Roles

In order to provide a framework for producing the draft standard the concept of roles was used, although it is not in the normative part of the draft standard (although it is in the rationale). The concept of roles helps in the explanation of the tasks but it is not rigorously defined and so could not be included in the normative part of the draft standard. This is just one example where the rigour of a draft standard conflicts with making it readable. Figure 1, shows the relationship of these roles, which are further discussed below. Note that this diagram is a simplified version of the one that appears in the rationale of the draft standard. The different roles may each take place on separate systems or combinations of roles may take place on one system.

Package Role

In the package role *developed software* is taken and put in a *distribution*. How the developed software got into a state to be packaged is outside the scope of the draft standard - the standard is not intended to cover the area of software development and source control.

Source Role

In the source role a distribution, or one or more components of it, is transferred to where it is to be installed. The transfer may also be to another source role - a staging operation. This transfer may take the form of electronic transmission or transfer on some medium. The concept of the source role came from some of the submissions which had very extensive functionality associated with this area. However, other implementations provided much less functionality and allowed for the role to effectively null in some circumstances.

Target Role

It is in the target role that the software is installed, that is it is deployed and manipulated to put it in a form that will enable it, eventually, to be run. One important aspect of the target role is that it may take place on a system which has a different architecture from that on which the software will run. Where there are discless clients the target role is taken by the server.

Client Role

In the client role the software is configured so that it will be in a state to be run. Configuration takes place on the architecture on which the software is to run. Installed software may be subject to being configured several times, for example comms software may be configured to serve several different paths. At mock ballot configuration was outside the scope of the standard because it was believed to be in the scope of another subgroup. However, many responses to the the mock ballot indicated that without configuration the draft standard would be incomplete and of significantly less use. Since the other subgroup had not progressed their work, configuration has been added.

Manager Role

Having the manager role provides for distributed control of the software administration process. The functions performed in the other roles can be controlled by the manager role. The manager role may be performed by the command line interface provided in the draft standard or by a management application. It is worth stressing again that the manager role can be on the same system as any (or all) of the other roles.

Developer Role

This role is specifically outside the scope of the standard. In the developer role software is *constructed* and placed into the form known as *developed software* which is the form in which it can be accepted by the package role. Typically this will involve activities such as compilation, source control, etc.

Structure of Packaging

The draft standard defines very precisely the format into which the software is packaged, this being the form that the source role transfers. The draft standard also defines the format for developed software, particularly the steering information which defines how to do the packaging. The draft standard does not define the format for installed or configured software (this being specific to particular architectures). The rest of this section gives the high level structure of the packaging format.

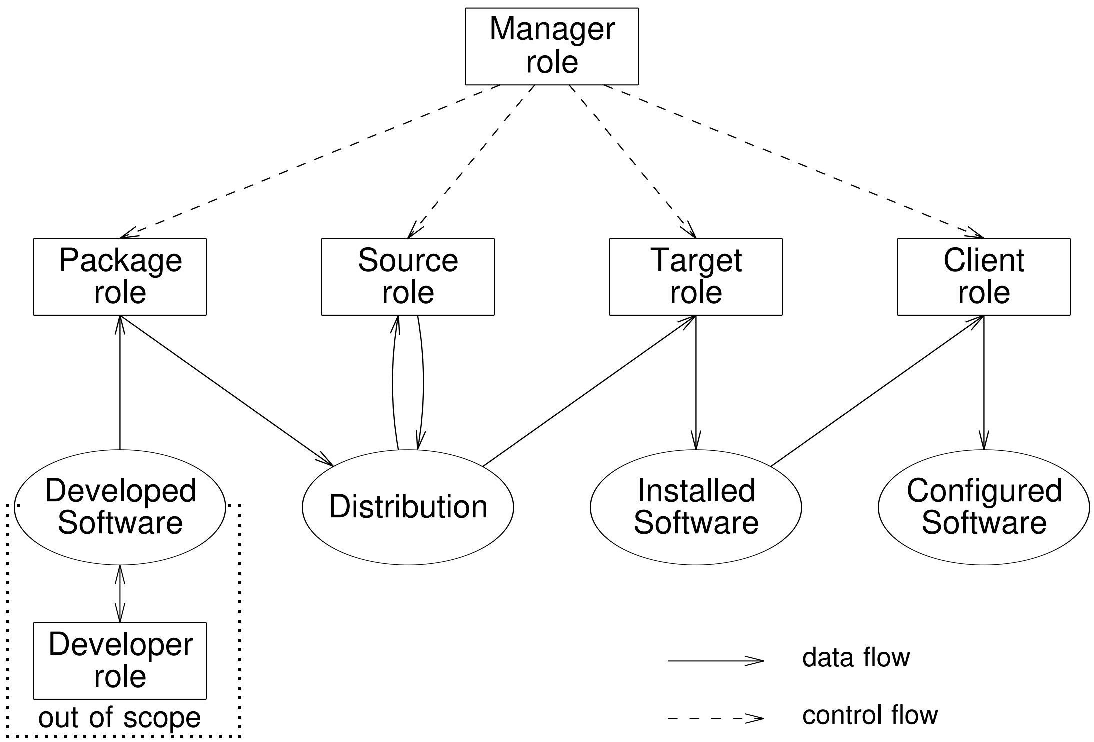

Figure 1: Roles in Software Administration

History

Although the various submissions showed considerable commonality in the fundamental concepts of the packaging format, this area of the draft standard caused significant problems, undergoing substantial changes before and after mock ballot. The problem lay with how many *levels* of structure there should be in the packaging format. *Products* containing *filesets* containing *files* was an obvious and simple format but one which all submitters had found to be inadequate. At one point (the first Santa Clara meeting) a recursive structure was proposed whereby a product could contain a product, and this could be to any depth. However, this was not what is commonly understood by the term "product". The group hence looked for some other (unloaded) word but ended up adopting, temporarily, the term RNC - for Recursive Notational Convenience!

However, although the recursive structure had many attractions, it was an unknown quantity, having no known existing practice. It was also felt that when work progressed to the detail, let alone implementation, there would be significant problems caused by this structure. An example might be dependencies (q.v.).

After many discussions over many meetings, the structure illustrated in Figure 2 has been adopted. This has subproducts within products and bundles within distributions.

Products

Although products were common to all submissions it took some effort to tightly define what they were since there were significant differences in the detail between the submissions. A late addition to the draft standard is the concept of *bundles* to group together products. These are explained below. Products are defined in the draft standard to have attributes like a name, a revision number, architecture, etc. One area of concern was that two software vendors might produce products with the same name. The draft standard already incorporated a mechanism to permit different versions of software to co-exist in a distribution and in installed software. However, in order for administrators to correctly identify a product, a vendor tag was added as an attribute that could be used to select a product. The group realised that there could still be a conflict if vendors used the same tag but felt it was beyond their remit to solve this problem. However, the administrator can also display the vendor description attribute where a vendor can put additional information, such as address, support telephone number, etc. There is probably little chance of this not being unique!

Products contain filesets and subproducts. Products can have dependencies on other products (see section on Dependencies).

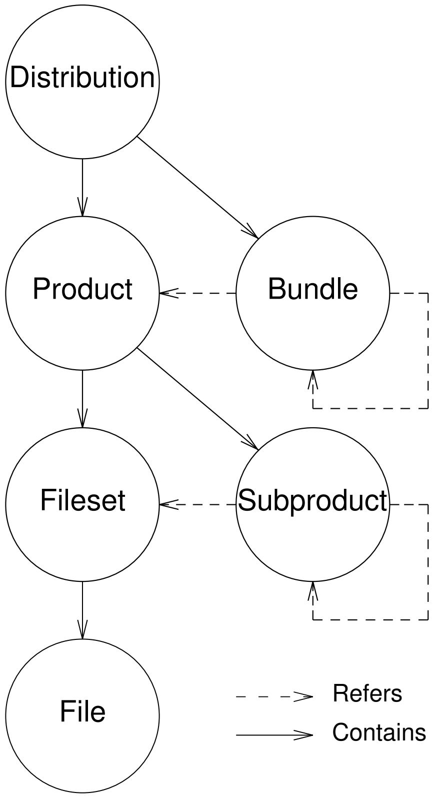

Figure 2: Structure of Packaging Layout

Filesets

A fileset is a collection of files that are logically related. The important point about a fileset is that it is the smallest unit that can be specified for the tasks defined in the draft standard. Filesets have many of the attributes of a product, such as name, version, architecture, etc. Filesets can have dependencies on other filesets, as well as bundles, products and subproducts (see section on Dependencies).

Subproducts

Subproducts are contained in products and are a method of addressing a group of filesets or subproducts. Hence a fileset (or subproduct) may be referred to from more than one subproduct. Subproducts do not contain anything and are not the recursive structure mentioned above. Subproducts are very simple and have few attributes (no revision or architecture, for example). A use of subproducts might be to group together the man pages, thus allowing an administrator to load a product, or products, but not the man pages for them.

Bundles

Bundles enable several products to be grouped and managed together. A major example of this was the operating system, which is a collection of products distributed as a whole. Bundles refer to products or other bundles in a similar manner to subproducts. They share many attributes in common with products. Products exist in a distribution in their own right; they do not have to be referred to from bundles. Some details of the operation of bundles is still being worked out by the group. There are discussions taking place about their attributes and the extent to which they still exist in installed software.

Packaging Information

The packaging format defines two types of information, the data that is the actual software (code, data, resources, etc.) and the control information that enables the installation process to take place. It is this control information that actually supplies the structure discussed in the preceding sections. In order to make installation efficient from a serial medium this information is required to be at the start of such a medium.

Tasks

In this section the tasks that are provided by the draft standard are described. These tasks are invoked using the CLI commands defined in the draft standard or by applications.

Phases

Tasks are implemented in three phases, the selection phase, the analysis phase and the execution phase.

Selection Phase

In the selection phase the filesets that are to be the subject of the task are determined. A fileset may be included because is has been specified individually or because it is part of a higher level component (e.g., a product). The way in which a selection is specified on the command line is covered in a later section. In addition a fileset may be included because it (or a component it is a part of) is needed to satisfy a dependency. In this case the fileset may be included without being specified to the task.

Analysis Phase

The analysis phase determines if the task is likely to succeed. This involves evaluating if there are enough resources, whether dependencies are satisfied, etc. Success in this phase does not guarantee that the task will succeed but failure should only occur if the task would certainly fail. A key aspect of this phase is that no change is made to the system so that if the phase fails part way through no recovery is necessary to revert to the initial state. The analysis phase is run for all selected products and filesets before proceeding to the next phase.

Execution Phase

In the execution phase the actual work of the task takes place, using the information from the selection phase.

Packaging

The task of packaging takes place in the packaging role and involves collecting the components of the software, together with control information, and making this into a distribution. The draft standard defines the way the steering information is supplied to the task as well as the way in which the component files are supplied. The information supplied to this task involves a detailed knowledge of the software and how it is constructed. It is envisaged that this task will be performed by the implementors of the software, either directly or as part of a make(1).

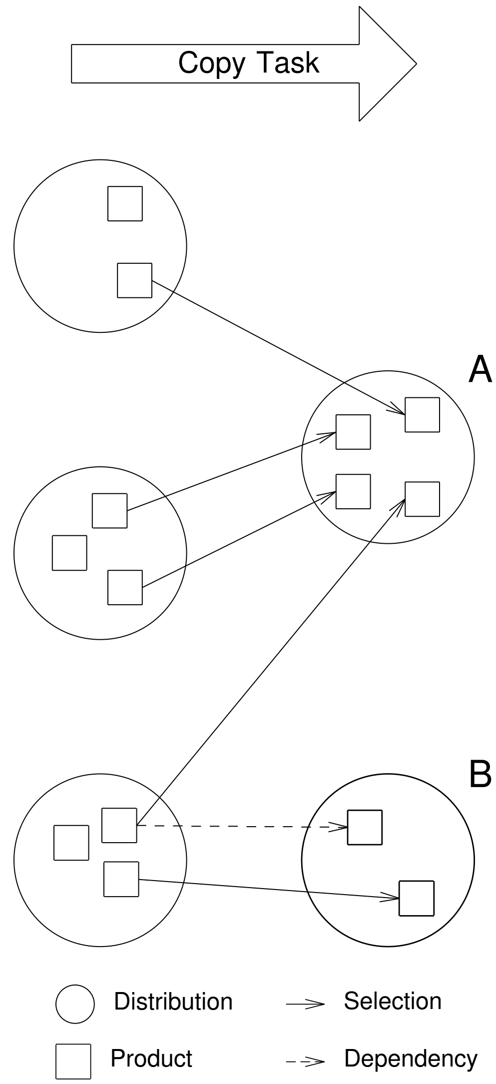

Figure 3: Example of Copy Tasks

Copying

Copying takes place in the source role and involves copying complete distributions or parts of them. Where parts of distributions are to be copied, the selection mechanism is used to define the components to be copied. Where the destination already exists, copying involves adding to the distribution. Copying may take place to or from different storage forms, for example copying to a serial medium.

It is envisaged that copying will be a common task performed by administrators. It might involve taking several distributions, received from the implementors or a software distributor, and constructing one or more further distributions from them. These new distributions may contain only parts of the original distributions, with only some products from bundles being copied or some subproducts from products. The latter case may occur where, for example, it was decided not to distribute the tutorial components of a product.

Figure 3 shows an example of creating two distributions, A and B. Distribution A is created by selecting 4 products from 3 distributions. Distribution B is created by selecting one product from a distribution and a second product is also copied because the selected product has a dependency on it. A similar example could show filesets or subproducts being selected from within products or products from within bundles.

Installing

Installation takes place in the target role and involves transforming software from the distribution format to the installed form in which the software can configured to be run. This involves operations such as creating directories, copying files, setting permissions, running scripts, etc. The installation process is explained in more detail in a later section. Input to the installation process may be a distribution in filestore (possibly copied from a serial medium) or directly from a serial medium.

Configuring

Configuring software is the final step before software is actually made operational and, unlike installation, always takes place in the client role and on the client architecture. The definition of configuration depends on the software but it is expected that it will normally be an operation that can be performed in significantly less time than the installation task. An example might be the installation of a new revision of a Message Transfer Agent. Configuring would specialise the software for the particular situation and make it the revision actually in use. Software may be configured more than once, each giving rise to a different configured instance. Taking the example of the MTA again, it may be that the software is configured for several different services. The parameters to configuration are specific to the software being configured, and are hence not part of the draft standard. They are supplied via the request task.

Removing

Removing a product involves deletion of the filestore elements that were created during the installation process as well as the management information relating to the product. There are some elements that are not removed, these being information that users would wish to have left. Examples of this would be the files that make up a database or the postbox in a Message Transfer Agent. These are identified specifically in the control information when the product is packaged.

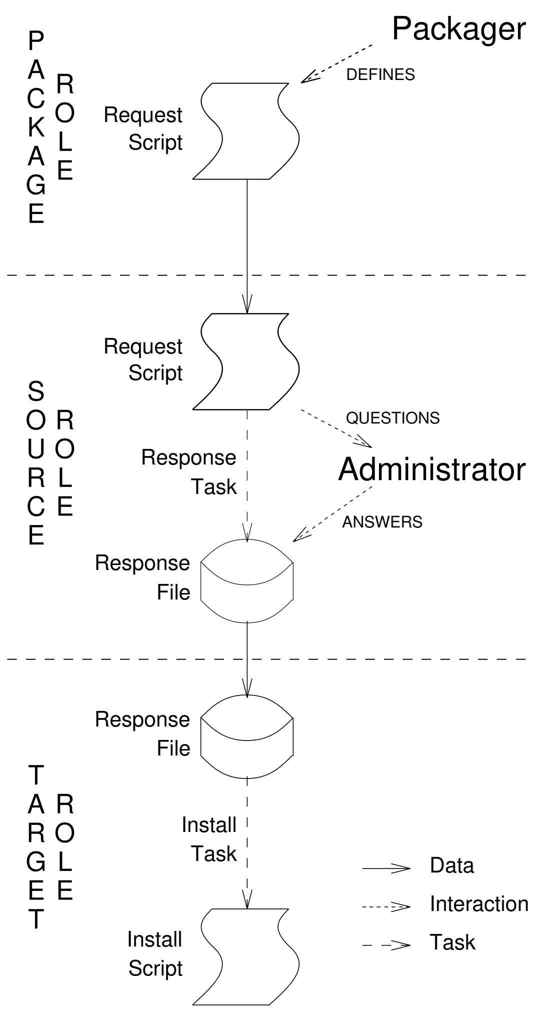

Figure 4: Request Task

Request

The installation and configuration tasks can involve the running of scripts defined during the packaging task. These scripts may need to obtain information to customise the work they do. If the scripts were to interrogate the administrator at the time the information was required, the installation or configuration task would be running interactively. To avoid this undesirable situation a script is defined during the packaging task which asks the questions. This script is run by the request task and the responses stored in a *response file*. When the installation or configuration task is taking place the scripts can use the information from the response file. The request task can be run entirely independently of the installation or configuration task and the response files distributed with the products to which they apply. If this is not done, the request task will be run at the start of the installation or configuration task. Figure 4 illustrates the use of the request script and response file.

Verifying

Verification of a distribution or installed software can be run in the source, target or client roles. It establishes the integrity of the information by checking the file attributes and checksum against the control information. Files that might change, and therefore should not be verified, are marked as such in the control information. If a customisation script (described in a later section) changes the contents of a file, the modification task should be used by the script to ensure that the management information is updated.

Listing

Listing can take place in the source, target or client roles and gives information about distributions and installed software. The selection process determines the items to be listed. The depth of information is given by an option.

Modification

Modification takes place in the source, target or client roles and is the process by which the management information is changed to reflect the information it refers to. This may be necessary because a customisation script has modified a file or because some of the management information associated with a product is inapplicable in a particular situation. Systems administrators who worked on the draft standard emphasised the importance of being able to correct the information, when (not if) it got out of step with reality. Since the way in which the management information is stored is implementation dependent it is necessary to provide a task to change it.

The Installation Process

One of the major items of the draft standard is how the installation of a product (or group of products) takes place. Installation of software involves those activities needed to transform it from the distribution to a state in which it can be run once it is configured.

Files

A fairly straightforward aspect of installation is the creation of directories to hold filesets and the copying of files from the distribution into the installed software. It is also possible to create links. The following sections discusses some of the more complex aspects of installation.

Dependencies

Dependencies provide an important way to ensure that software is correctly installed, configured, copied or removed. During the selection phase of a task a check is made for dependent software. If dependencies are not satisfied the task will fail (this can be overridden). A dependency is an attribute of a fileset that refers to a bundle, product, subproduct or other fileset. A dependency may also be an attribute of a product, which means that it applies to all filesets within the product.

Consideration was given to allowing dependences as attributes of subproducts but this was dropped because subproducts are references not "containers" and the rules would have been too complex.

The following sections describe the three types of dependency defined in the draft standard.

Prerequisites

A prerequisite must already have been installed before the software that depends on it is installed or it must be installed as part of the same installation task. During the selection phase of a task, products may be added to the selected set in order to satisfy prerequisite dependencies.

Corequisites

In the case of a corequisite, the software that is depended on must be installed and configured in order for the dependency to be satisfied. Dependencies such as this might occur for parts of the operating system.

Exrequisites

In this case installation cannot take place if the exrequisite has already been installed or has previously been selected during the install task. Dependencies such as this might occur where versions of a product cannot co-exist on a system.

Customisation Scripts

A common feature of all submissions was the use of scripts to allow installation and configuration to be customised. These scripts are defined during the packaging task. The scripts (apart from the configuration script) are run in the Target Role and thus not necessarily in the environment or on the architecture on which the software will be run. Environment variables for the scripts define the final environment. The method of returning information from scripts is also a problem and a totally satisfactory solution has yet to be found.

Scripts may be associated with products and with filesets. In principle each different fileset in a product could have a different script. Existing practice indicates that such a situation would be unusual and that product scripts are likely to be the most common. A exception to this might be filesets that make up the operating system.

Check Script

The check script is run during the analysis phase of the task to supplement the checks done automatically. For example, the automatic check for sufficient disc space could be supplemented by a disc space check that is dependent on some customisation of the installation specified in the response file. Since the scripts are executed during the analysis phase, they are not allowed to make any modifications to the target role.

Installation scripts

There are two installation scripts, the pre- and post-installation scripts run before and after the files are copied from the distribution. These scripts are run in the environment of the target role, not the client role. Examples of such scripts are the production of a new version of a product by applying changes to a previous revision and the transformation of data into a new format for a new revision of the product. Virtually the only constraint on these scripts is that if they modify the installed software a call to change the management information must be made (modification task). These extensive possibilities raise problems for the draft standard since it is difficult to ensure that the rules given are sufficient to guarantee interoperability. It is probably for this reason that so much discussion within the group concerned this aspect of the draft standard. To enable recovery to take place there are also *undo* scripts for pre- and post-install.

Removal Scripts

Like the installation scripts there are pre- and post-removal scripts. The draft standard does not define what the removal scripts should do except that they should reverse any changes that the installation scripts have made and which have not, or could not, be reflected in the management information. Hence if an installation script creates a file and adds this to the management information such a file will be deleted automatically. However, if a data file needs to be transformed into a different format that will have to be handled by the removal script.

Configuration Script

These scripts perform functions that must take place on the architecture on which the software will run or which are associated with the configuration of a particular instance of the software. Since the only substantive action defined in the draft standard for the configuration task is the running of the configuration script, a product can only effectively be configured if such a script is supplied. The parameters to the configuration task are supplied to the configuration script by means of the request task. Examples of configuration scripts are a compilation to the architecture of the client role or the definition of particular services.

Product Location

The packaging layout specifies a default location in the filestore where a product will be installed. This can be overridden by an option to the task.

Simultaneous Versions

It is possible to install different versions of a product simultaneously, provided the product can be installed anywhere in the filestore hierarchy (i.e., it is relocatable). The version of a product includes its revision and the architecture it is to run on. Hence, it is possible to have simultaneous installation of multiple revisions of a product as well as installing versions for different architectures (important for servers of discless clients). Depending on the product, it may or may not be possible to configure multiple revisions simultaneously.

Overlaying

Only one product can exist at one location. If an attempt is made to install another product (or another version of the same product) at the same location it will either be rejected as an error or the original product will be deleted. The action to be taken can be selected by an option to the task. It is expected that all products will be relocatable and the installation of a new version of a product will not be done by overlaying.

Recovery

Recovery is the process of undoing the effect of a failed task, addressed here in terms of installation but also applying to copying and, to a lesser extent, packaging and configuring. Recovery is only significant when a product has been overlaid. Where a new version is installed simultaneously with an old version, recovery merely involves removing the partially installed new version. As has been stated, recovery was not addressed in the draft that was circulated for mock ballot. This was because the discussions up to that point had not produced a consensus on what should be done. However, responses to the mock ballot showed that recovery would have to be addressed in the final balloting draft. In the event of a failure there are basically two choices, to delete what has already been installed or to leave what has been done so that a subsequent installation does not have to re-install parts already successfully installed. The choice of these could be an installation option. The following sections discuss some of issues involved.

Overlaid Products

Information was provided to the group about implementations that provided recovery by roll-back or by copying and deletion of the old version. Whatever the implementation there are implications in terms of storage required, already a potential problem area if both the distribution and installed software were present on a system.

Administrative Applications

Applications that provide an advanced interface to Software Administration would handle recovery in their own style. In order to enable this to happen the distributed interface would provide detailed control over the phases of the installation process (events on completion of a phase and control over the transition between phases). Any facilities in the draft standard must therefore cover the requirements of the command line as well as administrative applications.

Level of recovery

The components selected for installation may be the result of a high level definition ("install this bundle") or a low level definition ("install these filesets"). It might be deemed necessary for the recovery action to be different in the two cases - and all the cases in between and combinations. However, this seems to imply that the draft standard should contain a very complex definition, detailing what should happen in each case, and providing equally complex overrides for the default actions.

Scripts

When an installation fails it is necessary to run scripts to undo the changes made by the installation script(s). However, it would be difficult for an implementor to ensure that such scripts would work irrespective of the the type or position of the failure.

Current Situation

The current proposal being worked on in the draft standard provides a fairly straightforward recovery mechanism. It is applicable to the situation where a product is overlaid and requires that, in the event of a failure, the product is restored to its original state. Two new scripts are proposed, the unpreinstall and the unpost-install scripts which undo any changes made by the corresponding install scripts.

Interactions During Tasks

One area where there was not commonality in the submissions was the facility for the installation scripts to ask questions of the task submitter. Since making the installation process interactive is undesirable some submissions effectively forbade any such questions whereas others enabled the questions to be answered at the start of the installation task and even allowed the answers to be distributed with the software. The draft standard adopts this latter approach - see the request task.

Software Service

Software Service is the term adopted to describe modifications made to product other than replacement by a different version of the product. This includes replacement of one or more files and in situ modification of the data within a file, the classic form of "patching". When this was initially considered many different methods of achieving it were described. However, there was no common core that could be discerned in these methods and the submitters were frequently not enthusiastic about their own methods. It was therefore difficult for the group to select an existing practice to standardise and for this reason it was omitted entirely from the mock ballot version of the draft standard. The rest of this section describes some of the issues and the current state of the draft standard.

Level Identification

One of the major topics for Software Service was the identification of the modifications that had been applied. In the completely general case each modification would be separately identified and the list of modifications would be available as an attribute of the modified product. However, this does not answer the question of how a task could check that a new modification was appropriate for the existing modification level of a product. Various schemes were in current use, from those that re-issued all previous modifications with each new modification to those that left it up to the administrator to select the modifications to be applied, handling any dependencies or exclusions between them.

Reversion

It is obviously necessary to be able to remove a modification from a product and in the general case this would either require a roll forward from the original, unmodified, instance of the product or would need roll back information to be kept.

Management Information

Modification of part of a product requires that the management information be updated. This would then enable the verify task to operate correctly and not report an error with respect to a modified product. With a roll back provision for reversion (see above) the modified management information would have to form part of the roll back log.

Current State of Standard

The current state of the draft standard is that there will be no additional facilities provided specifically for software service, although the rationale will explain how it can be achieved. This involves the overlaying of one fileset with a new version that has one or more of the files changed. The installation scripts can be used to provide roll back, identification and dependency checking. This is the only solution that seemed capable of accommodating the diverse schemes currently in use.

Installing the Operating System

While the draft standard does not address all aspects of operating system update and initial installation, it does provide the basic functionality so that it can be used as a fundamental part of these processes. Facilities provided include marking files as being part of the operating system and indicating that a product or fileset will not become effective until a re-boot occurs. Excluded, however, are the final stages of switching from the old to the new version of the operating system, which would take place during the configuration stage. The initial installation of the operating system on an empty system requires special techniques since services that are normally assumed to be present (e.g., the filestore) are not available. The draft standard only deals with installation of software when a POSIX compliant operating system is present and so is not applicable to the initial installation of the operating system until this is true. This does not, of course, preclude a vendor from providing such facilities but they would be extensions to the standard.

Tasks using the CLI

One of the objectives for the draft standard has been stated as *Operator Portability*, meaning that, on any system that complies with the draft standard, an administrator would find a well known set of Commands with which to perform software administration tasks. Nevertheless, it was recognised that the interface to software administration, and particularly distributed software administration, would increasingly be the province of an integrated interface, particularly one based on a Graphical User Interface. Such an interface would have a significant advantage where software was being distributed to, and installed on multiple machines simultaneously, a task which is inherently asynchronous. Several of the submissions indicated that such implementations already existed.

The Command Line Format

All submissions provided commands to invoke the tasks and the draft standard was based on these. The basic form of a command is

```
command [options...] selections \
                    [@ target ...]
```

meaning the the command operates on the software identified by *selections* and the tasks take place on the hosts specified by *target*. The format of the options and target is covered in the rest of this section. The selections, being a significant issue in their own right, are covered in another section. The commands implement the tasks already defined.

Options

An important issue that had to be addressed was the sheer number of options that had to be accommodated. Not all options from all submissions were included but there was an inclination to accept that if a facility had been found necessary or useful it should be included. This issue of the number of options had already been addressed by the sub-group working on the Print standard, P1003.7.1, and a compatible approach was adopted. This involved specifying options in a quoted string given as the -x option to the command, or in a file, the pathname of which is specified in the -X option. Within the quoted string, or file, an option would consist of an identifier and a value. The identifier consists of lower case letters and underscores. These identifiers are not localisable to other languages.

Host Definitions

The format for specifying the machine on which the task is performed is

```
@ target...
```

and this was generally liked as being intuitive although it does not have any applicable precedence as a separator of operands (its use in mail aliases and Berkeley commands is different). This syntax does not appear in POSIX.2 but is legal according to the utility guidelines of that standard. As distributed utilities extend the problem space that POSIX.2 addresses, avoiding extensions was not deemed to be essential. In the end, the decision of the working group was that the use of @ was acceptable, and indeed desirable over alternatives such as moving the operand to the options.

Selections

A selection defines the items that are the subject of an operation, for example a selection might define the software products that are to be installed from a distribution. At the simplest level this would just be the name of a product. However, there were several areas where the selection got more complex and there was a struggle to achieve the necessary flexibility without a grossly complex syntax. The following sections describe the details of the selection and the objectives that were being addressed.

Depth

A selection can specify bundles, products, subproducts or filesets and so can be as specific or general as required. The implication is always that all the components of the item specified are selected.

Versions

In the draft standard, the product's version is an attribute that identifies its intended architecture, vendor, and revision of the product itself. Hence the same revision of a product can have several versions, each for a different combination of hardware and operating system. The specification of the architecture in the selection provides wild cards and the comparison of the revision takes into account the common dot format, e.g., 2.03.

Locations

A selection may also specify the location where the product is to be located as a result of the operation, overriding the default in the product. For example, for the install task the location would define where the product is to be installed. This feature of a selection is a bit of an oddity because the rest of the selection is concerned with the source of the operation whereas the location is concerned with the destination. However, it is necessary because there may be several selections each needing to be located in a different place.

Dependencies

Selections are also used for dependencies, that is for references from a product or fileset to a bundle, product, subproduct or fileset, but in this case the location cannot be specified.

Customisation

The systems administrators who had participated in the development of the draft standard had emphasised the importance of avoiding fixed restrictions whilst at the same time enabling defaults and limits to be set for any particular installation. The existing practice supported this concept and and so this facility was built into the draft standard.

System Wide Defaults

On any system there will be one defaults file which gives the defaults for about 25 aspects of software administration. In addition different defaults can be specified for different tasks. So, for example, the default for whether to try to automatically resolve dependencies could be set to **true** for installation but **false** for copying a distribution.

Local Defaults

The system wide defaults can be over-ridden by the options file to a particular command, a file which is in a similar format to the system wide defaults file. These in their turn can be over-ridden by what is specified on the command line.

The Software Catalogue

The term catalogue applies to a distribution or to a collection of installed software. Most of what the draft standard defines about a catalogue is the control information, which is actually very similar between the two. The difference is that the format of a distribution is defined by the draft standard whereas the format for the catalog for installed software is not. In this latter case it is implementation defined how the catalog is stored although the draft standard does define standard ways of accessing it. For example, the list task reads it and the modify task changes it.

Contents of a Catalogue

Information in the catalogue defines the contents (bundle, product, etc.) of a distribution or installed software, giving the attributes of the components (name, revision and dependencies for example).

Multiple Catalogues

A valuable contribution from Systems Administrators in the group was the need for multiple catalogues, for example corresponding to development software, software under test and production software. The draft standard hence allows for the catalogue to be specified as part of the syntax of the commands. This does however raise the question of how one might be able to find all the catalogues on a particular system. It would be a distinct advantage if this could be achieved in some way but so far this has not been incorporated in the draft standard.

Filestore Structure

The draft standard is based on a POSIX compliant filestore structure but does not specify any other detail about how installed software should be mapped other than that there must be a node under which the product files are installed. This requirement does not exclude the possibility of some files being located elsewhere although this is discouraged.

Software Layout

Software should be constructed so that it can be installed relative to any point in the filestore hierarchy. This is particularly important for the simultaneous installation of multiple versions of the same product. However there are some types of software for which this is not possible, particularly the operating system itself. In such circumstances the software will have to be constructed to provide some other method of handling simultaneous versions, possibly by some special action as part of configuration.

Alternative Root

Sometimes it is necessary to install software relative to a virtual (or alternative) root. This means that absolute references in the installation to the filestore hierarchy are taken to be relative to the alternative root. This is particularly useful for installing operating system software for a discless client or on a disc unit that will be installed in another system (preloaded software). The discless client example is illustrated in Figure 5 in which software is installed on the target with node D as the alternative root. For the client J *is* the root, node K is node E, etc. and so it appears to the client as if the software had been installed with the actual root as J.

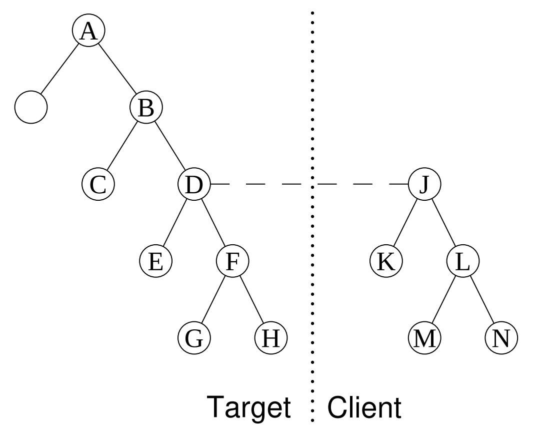

Figure 5: Filestore Structure for Discless Clients

Discless Clients

For discless clients the alternative root facility is obviously important, particularly for the operating system. Installation of other software only requires that the correct location in the (server) filestore is chosen for the software to be visible to the client. However, if the management information is to be visible to the client it is important that this too is located in the correct place. The provision of multiple catalogues is hence an important facility for discless clients.

Heterogeneous Management

The group would very much liked to have made the standard yield implementations that were interoperable at the task level. That is to say that the manager role could manage any of the other roles irrespective of the systems on which the roles were implemented provided they conformed to the standard. This would provide not only the capability of heterogeneous management using the commands defined in the standard but also a mechanism for enabling management applications to be written which could manage conformant systems. Unfortunately this would require the standard to refer to some mechanism for performing distributed tasks and no such mechanism is available as a *formal* standard. However, the group did receive several submissions specifying how this could be achieved using *de facto* standards. In addition some work was done on the formal definition of Managed Objects that corresponded to the definitions in the standard. An agreement has been reached with the X/Open Systems Management Working Group that they would progress this aspect of the standard, to be published in due course as an X/Open Specification.

How to Participate in the Ballot

When the ballot is about to take place (expected to be April/May 1994) the IEEE will advertise for participants. Anyone can submit comments but only those from members of the IEEE (or the Computer Society of the IEEE) are counted for the ballot; comments from others are "for information only". To ensure that you are notified of the ballot send your details to the author or the chair of the group, Jay Ashford at ashford@austin.ibm.com.

Acknowledgements

A lot of people have contributed to the draft standard, too many to be mentioned here. However, particular thanks are due to Jay Ashford, Matt Wicks and George Williams who have reviewed this paper to ensure that it reflects what the draft standard actually says rather than my own prejudices.

Author Information

Barrie Archer is a Systems Designer working in ICL Client-Server Systems. He works on the strategy of ICL's Systems Management products and participated in the development of ICL's OPEN*framework* Architecture for Systems Management. He is the ICL representative on the X/Open Systems Management Working Group and POSIX 1003.7 Working Group. He can be reached by mail on ICL Lovelace Road, BRACKNELL Berks, RG12 8SN, UK and electronically at barcher@oasis.icl.co.uk.