# A UNIX-based System for Software Configuration Management

D. MACKAY, G. BALL, M. CROWE, M. HUGHES, D. JENKINS AND C. NICOL*
*Software Tools Research Group, Paisley College of Technology*

***This paper describes an approach to a project management which is sufficiently flexible to allow development methodologies and objectives to vary from one project to another. Enhancements to the UNIX C library are described which allow all the usual UNIX tools to support hierarchical file attributes, version control and controlled access, by using this new library. Some additional utilities are provided to assist in project management. These have privileged access rights, and their use is subject to restrictions imposed by a project schema.***

*Received August 1984*

## 1. INTRODUCTION

Software configuration management (SCM) is concerned with the control of changes to each of the many items generated during the life cycle of a software product.[1, 2] A large amount of information is produced during development which must be carefully identified and protected from unauthorised change, otherwise the product will rapidly become uncontrollable. Configuration management is a vital activity which must be performed throughout the life cycle from project initiation to release and evolution.

SCM is, without adequate computer-aided support, a laborious and time-consuming task which is often neglected by software developers. However, recently interest has been aroused in software tools to assist the activities of configuration management. The Department of Trade and Industry STARTS guide lists a number of currently available tools which offer support in this area.[3] The STARTS guide lists thirteen functional requirements for configuration management tools, but is unable to identify any system which meets all of these requirements.

It is essential to any successful approach to SCM that no software tool violates any constraints imposed. However, there are a number of competing methodologies for software development, each of which has its own SCM procedures, and at first sight it would appear that a new software toolset is required for each new methodology, however tentative.

This paper suggests an approach to SCM which enables the popular software toolset provided in the UNIX operating system to support a variety of SCM procedures. A number of features common to many SCM systems are identified, and supported in the UNIX C library, in such a way as to provide an SCM-oriented file system with versions and controlled access. The UNIX toolset is then re-linked to use this library in place of the standard C library. These tools then will respect the procedures specified on a per-project basis by the project schema, which also controls the use of some privileged SCM utilities. The resulting environment is called the Project Development Environment (PDE).

This work forms part of a three-year project begun in September 1983, funded by the Science and Engineering Research Council.

* Contact: Dr M. K. Crowe, Paisley College of Technology, High Street, Paisley PA1 2BE.

## 2. CONFIGURATION MANAGEMENT UNDER UNIX

UNIX is a general-purpose operating system in widespread use, particularly in the academic and research communities.[4] The file system is hierarchical, allowing users to create directories containing files and possibly other directories. Many of the UNIX commands can operate recursively, allowing entire subtrees to be listed, copied or deleted. The result is a flexible and convenient system in which to develop software. However, UNIX is deficient in support of SCM, failing to meet the requirements given, for example, by McDermid and Ripken.[5] In particular:

(1) It is not possible to have several versions of files other than by making copies with different names.

(2) Relationships between files cannot be recorded except by using ad hoc methods.

(3) The access control mechanism is not sufficient. For example, to permit a user to delete a file from a directory, it is necessary to grant permission allowing any file in that directory to be deleted.

The PDE system, developed at Paisley College, extends UNIX by providing a set of tools and an enhancement to the file system to overcome these problems.[6] In order to meet the demand for SCM, a number of tools, such as the Source Code Control and Revision Control Systems, have been provided under UNIX.[7, 8]

The Source Code Control System is a tool for storing and controlling changes to text files.[7] In SCCS all versions of a file are stored as one file by recording only the differences between successive versions. Versions are protected from unauthorised access and modification. SCCS has been widely and successfully used by many projects. However, it has a number of disadvantages, as follows.

(1) When a version is required it must be extracted from the archive, and the resulting file is no longer under the control of SCCS.

(2) It is not possible to identify configurations of files (i.e. hierarchies) and place these under the control of SCCS.

(3) It is not possible to have a default version other than the most recent.

The Revision Control System,[8] a more recent tool, is similar to SCCS but offers some improved facilities. Both of these systems are useful tools for managing revisions to text files. However, recent surveys have shown them

to be inadequate for comprehensive configuration management.[3]

## 3. THE PROJECT DEVELOPMENT ENVIRONMENT

The Project Development Environment (PDE) has been developed by the Software Tools Research Group at Paisley College with SERC funding.[6] The system supports the idea of a project schema which allows management to control and restrict operations on the hierarchies under their control. The PDE's facilities are provided at the system call interface, which allows the existing UNIX tools to be integrated with the PDE, in most cases without modification. No change is required to the UNIX kernel.

In the following discussion the term *object* refers to an ordinary UNIX file or a directory.

### 3.1 Attributes

In UNIX the directory objects provide a mapping between file names and file objects. A file name is one of the attributes of a file object. In addition each file object also has a number of other attributes. These attributes are recorded with the file name and include: the owner of the file, access permissions and the last access time.

The PDE extends the facilities of UNIX by allowing users to associate additional attributes with files subject to per-project SCM requirements. Functions are provided, in a similar way to UNIX system calls to set an attribute of a file, obtain the value of an attribute, obtain all the attributes of a file, and delete an attribute. Many of these functions are also available at command level as utility programs. The user may access attributes using a utility, ATT, provided by the PDE or by using a modified version of the command interpreter, the shell. A utility, FINDATTR, is provided to search part of the file hierarchy for a given attribute.

For the purposes of defining the SCM requirements of the project for an object it is desirable that some requirements are inherited by the sub-objects of an object. Thus an object is said to have property X if X is an attribute of the object or one of its ancestors. Properties are used by the controlled access mechanism, and in controlling the use of some privileged SCM utilities. The collection of attributes and properties for a project is referred to in this paper as a project schema, although the relationship with a database schema is somewhat tenuous.

A small set of attributes known as *hardwired attributes* are built in to the system for specific uses. These are as follows.

ACCESS
: This attribute augments the access controls provided by UNIX, and is the basis of the controlled access facilities, described later.

ADMINISTRATOR
: The name of the maintainer of the project schema.

ARCHIVED
: For objects archived by the ARV program, this attribute contains the name of the user who archived the object, and the data and time of the operation.

SYS_LOG
: Also used by the controlled access facilities, this attribute specifies the name of a file in which to place log messages.

PRE_⟨tool name⟩
POST_⟨tool name⟩
: Used in the selection of pre- and post-execution programs.

### 3.2 Privileged SCM Utilities

The privileged utilities allow users to make controlled changes to controlled objects. They have three phases of operation: pre-execution, and post-execution.

The pre-execution phase checks that the requested operation is permited, with respect to the project schema. The post-execution phase records the successful completion of the utility. This may involve setting one or more attributes, making a log entry, or sending mail to some user. It may also take appropriate action if the utility fails to complete successfully, such as restoring the previous state of the system: this is possible since the actions of the privileged utility are known to the pre-execution program.

The functions of the pre- and post-execution phases reflect the particular configuration management procedures being followed in the development of a project. Since these functions may vary from project to project they are not coded as part of the tools themselves.

The actions to be performed are defined by *pre-execution* and *post-execution* programs which are specified by the project schema. These programs will usually be written as command scripts.

There are five privileged utilities as follows.

(1) ATT: update the attributes of an object.

(2) MKV: make a version of an object.

(3) DEV: establish a version as the default version of that object.

(4) REV: place an object under control.

(5) ARV: archive/restore a version.

It is intended that project administrators will extend the facilities of the basic environment by adapting the pre- and post-execution programs and adding more specialised tools, according to their requirements.

### 3.3 Version Control

In UNIX it is only possible to have several versions of a file by creating copies with different names. There is no way of recording relationships between these copies.

The version control facilities of the PDE have the following features.

(1) Several versions may simultaneously be available as members of a *family* of versions under one name. Individual versions may be selected by appending a *version identifier* to the family name.

(2) A *default* version is available which is assumed if no identifier is given. This need not be the most recent version.

(3) Entire hierarchies can exist as a family of versions, not just individual files. This is an important advance on SCCS and RCS.

(4) No fixed version numbering or naming scheme is enforced.

The program MKV is used to make a new version of

an object. The object may be a simple file or a configuration object consisting of a directory and its sub-objects. MKV takes two arguments: the name of the object and the identifier of the new version to be created. When MKV is initially applied to an object it initialises the version family for the object.

The user may archive versions using the ARV program, which employs RCS to create and maintain the necessary difference files.[9] The ARV program works on both configuration objects and simple files and may be used to unarchive any object which was previously archived.

### 3.4 Controlled Access

UNIX provides read, write and execute (or search) permissions for files and directories. The PDE augments these basic facilities by providing a *controlled access* mechanism implemented by the use of the ACCESS property. Five additional types are provided.

(1) Append to file, or create entry for a directory.
(2) Overwrite file, or delete entry for a directory.
(3) Delete object.
(4) Alter object attributes.
(5) Change default version.

The ACCESS property has a value consisting of a five-character string each of which is one of the following: y, allow access; n, disallow access; l, allow, but log access.

The PDE provides a program, REV, which may be used to place an object under control. REV achieves this by changing the ownership of the object, changing the UNIX file protection modes and setting the ACCESS attribute such that all types of access are disallowed. The pre-execution program for REV will normally consult the attributes of the object to check that it conforms to the schema.

## 4. IMPLEMENTATION

It was considered extremely important that the process of adapting the standard UNIX toolset for the PDE should be as straightforward as possible. This was achieved by implementing the facilities of version selection and attribute management by a set of low-level subroutines known as the *PDE kernel*, which are called from the system call subroutines in the C library. This kernel provides version selection and attribute management.

The attribute management routines are available to all UNIX programs as standard library routines. The activities of version selection are hidden from the user, but form part of the more general function which maps a PDE object name on to that of the corresponding UNIX file or directory.

When within the PDE, the mapping function will be invoked every time an object is accessed via one of the UNIX system calls, for example those which are used to open, close, rename and delete files and directories. The existing system calls have been modified to do this, but take no action if the request is made from outside the PDE may refer to PDE objects in the same way as more traditional UNIX programs refer to files and directories. Moreover, most existing UNIX programs may be integrated with the PDE simply by incorporating the modified system calls in place of the existing ones. This is accomplished by re-linking the programs with the new C library containing the modified system calls.

The PDE kernel functions make use of a table associated with each directory, and stored as a hidden file in the directory. This table contains, for each sub-object, the attributes of the object, along with version information. It augments the information about an object which is held in the UNIX directory. The mapping table is locked during PDE updates, and a system of timestamps prevents conflicting updates. Updates occur during system calls such as CREAT or UNLINK and are carried out by the PDE kernel. If an object has versions, an extra UNIX directory is inserted which is invisible to the user. The actual UNIX path names of PDE objects consist of anonymous digit strings selected by the PDE kernel.

This table can be compared to the Common Apse Interface Set[10] notion of a *node* as a carrier of information about an object. CAIS provides specifications for a set of Ada packages designed to promote portability of Ada development tools, and other programs using the APSE. An earlier paper[11] has shown how the PDE could support this interface. An Ada interface has been provided to the existing facilities and is described in.[12]

Experience with the current implementation has not shown any appreciable slowing down of the system. However, there is obviously some overhead associated with version selection, and any program that needs the PDE kernel is appreciably larger in size. A tempting development would be to incorporate the PDE kernel into UNIX by creating a new layer in the operating system.

## 5. EXAMPLE

The following command scripts are examples of pre- and post-execution programs for REV, the tool which places an object under strict change control.

```
# Pre-execution for REV.
# Check for the existence of certain attributes,
# signalling failure if not present.
if attr -n $1 TESTED
then
  echo 'Object not tested, cannot control'
  exit 0
fi
if attr -n $1 CONTROLLABLE
then
  echo 'Object not controllable'
  exit 0
fi
exit 1
# Post-execution for REV.
# Set an attribute and make log entry.
att $1 CONTROLLED
echo -n $1 'controlled on': >> /usr/pde/log
date >> /usr/pde/log
```

The schema for a project is specified by giving the pre- and post-execution programs for a project, and simply setting these in the attribute list of a given object (probably a directory), along with the name of the

ADMINISTRATOR. A high-level language for specifying the schema, so that the pre- and post-execution programs are generated automatically, is an interesting possibility that has not yet been tackled.

It is crucial to the PDE concept that all normal software development tasks proceed using the standard UNIX toolset, which will operate normally except for the object-naming conventions and version selection. Apart from the shell, and some file-oriented utilities such as rm and ls, no change to UNIX utilities is required.

The small set of privileged SCM utilities provided may be extended if required, through use of the PDE system calls. A privileged SCM utility would run in 'setuid-root' mode, though the pre- and post-execution programs would not. For reasons of space, it is not possible to print the source text of any of the privileged utilities described above (which are written in Pascal). But the following C source of a function to get a property may be of interest. It uses two of the PDE system calls, PDE_pname and PDE_gattr.

```
# include <pde/attr.h>

char *
get_property(p_name,o_name)
char *p_name, *o_name;

/* Return value of property <p_name> of object
<0_name> */

{
  char *p_val;
  char *fullname, *np;
  int p_stat;

  /* Calculate primary pathname of object */
  if (!(fullname = PDE_pname(o_name))){
    perror(o_name);
    return (char *)0;
  }

  p_stat = PDE_gattr(fullname,p_name,&p_val);
  while (!(p_stat == SET || p_stat == VALUED)){
    if (p_stat < 0){
      /* error in calculation */
      perror(o_name);
      return (char *)0;
    }

    /* trim last component off name */
    np = fullname+strlen(fullname);
    while (*np != '/')
      np--;
    if ((np-1) == fullname)
      /* at start of name-property not set */
      return (char *)0;
    *np = '\0'; /* mark new end of name */
    p_stat = PDE_gattr(fullname,p_name,&p_val);
  }

  /* property must be set if we get here */
  return ((p_stat == VALUED)?p-val:(char *)0);
}
```

## 6. CONCLUSIONS

An extension to UNIX has been described which provides better configuration management facilities in a flexible way. These facilities are provided at system call level, allowing the existing tools to benefit from their advantages.

The system as described above is under evaluation at two sites, and is available to other participants in the Alvey Programme. Continuing development work on the PDE will be concerned with providing typing of file attributes, a project schema definition language, and further investigations into the implementation of the CAIS specifications. In addition, a comparative study is being made of the PDE and other configuration management tools.

## REFERENCES


1. E. H. Bersoff, V. D. Henderson and S. G. Siegel, *Software Configuration Management*, Prentice-Hall, Englewood Cliffs, N.J. (1981).
2. J. K. Buckle, *Software Configuration Management*, Macmillan, London (1982).
3. Department of Trade and Industry, *Software Tools for Application to Large Real Time Systems* (*STARTS*). H.M.S.O. (1984).
4. D. M. Ritchie and K. Thompson, The Unix Time Sharing System. *The Bell System Technical Journal* **56** (6), 1904–1929 (1978).
5. J. McDermid and K. Ripken, *Life Cycle Support in the Ada Environment*. Cambridge University Press (1984).
6. D. Mackay, The Project Development Environment: Specification, Software Tools Research Group – Internal Report No. 6, Paisley College of Technology, Paisley, Scotland (1984).
7. M. J. Rochkind, The source code control system. *IEEE Transactions on Software Engineering* **SE-1** (4), 364–370 (1975).
8. W. F. Tichy, Design implementation and evaluation of a revision control system. *Proceedings of the 6th International Conference on Software Engineering, IEEE, Tokyo* (1982).
9. M. Hughes, Arv – an online archiver for the PDE, Software Tools Research Group – Internal Report No. 5, Paisley College of Technology, Paisley, Scotland (1984).
10. U.S. Department of Defense and KIT/KITIA CAIS Working Group for the Ada Joint Program Office, Draft Specification of the Common APSE Interface Set, Version 1.2, *CAIS* (1984).
11. M. K. Crowe *et al.* Supporting the CAIS from UNIX. *Ada UK News* **5** (1), 48–50 (1984).
12. D. Mackay, Providing an Ada interface to the PDE, Software Tools Research Group, Paisley College of Technology, Paisley, Scotland (1985).