Version and Configuration Management on a Software Engineering Database.

Ian Thomas *

GIE Emeraude, Bull SA, 68 Route de Versailles, 78430 Louveciennes, France

1. Introduction.

The PCTE interfaces provide a set of facilities to builders of software engineering environments and integrated toolsets. Part of the interface definition describes a data repository, the Object Management System (OMS), intended to support the storage of all of the information necessary for the software development process.

The OMS data model is based on the binary Entity-Relationship model. It offers objects, links and attributes, all of which are typed. Object types are organised in a hierarchy with inheritance of attributes and relationships defined on an object type to its descendants. Relationships are bi-directional associations between objects. They can be considered as a pair of mutually inverse links. See [BOUD88] for a brief description of PCTE.

2. Version and Configuration Management on the OMS.

Over the last few years, there has been an evolution from the use of file systems as data repositories for SEEs to the use of data/object bases. PCTE's OMS exemplifies this evolution. The additional richness of the data models of these data/object bases allows explicit representation of relations between the objects manipulated in the SEE, including dependency information of interest for Configuration Management.

The OMS data model is sufficiently general to allow the modelling of versions in its schema. For example, one could represent a source file as an object and all versions of the source file as separate objects linked to the source file object by links of a special type.

It seems desirable to avoid explicit representation of version organisation in this way. The reasons for this are that it is extremely difficult to achieve consensus on a common version organisation and many organisations will develop their own or adapt standard ones. Since tools need the schema to access objects and therefore need to know the version organisation used within the schema in order to access versions of objects, these tools would have to vary slightly from organisation to organisation. At the very least, they would have to have all navigations (pathnames) re-interpreted.

We are therefore seeking maximum independence of tools from version organisation in the interests of tool portability (although this does not apply to tools that need to know about version organisation such as CM tools). Ideally, a tool would have no knowledge of versions but would have its own defined data model, suited to its functions and operate using only that model. We call this independence from version organisation **transparency**. Version organisation and access to appropriate elements by a tool is transparent to a tool.

There have been some systems that have provided transparency of various sorts. Apollo's DSEE [LEBL85] provides transparency by separating the description of the components of a configuration and what has to be done to rebuild it, from the description of the particular versions of components that have to be used in a particular build operation. References by a tool executed during the build (e.g. *open*) to components are translated by the operating system to refer to the appropriate version bound to that component. The tools that are executed during the build are therefore ignorant of the existence of several versions - they can operate in a world in which there is only one version of the components that interest them.

Sun's NSE environment concept offers the possibility of creating a new work context within which a user and tools can operate independently of whether versions of objects in that environment exist in other environments.

* Author's present address: Hewlett-Packard, Software Engineering Systems Division, 3500 Deer Creek Rd., Palo Alto, CA 94304

An approach involving transparency had already been adopted by CADES in the mid-seventies [McGU79]. CADES managed a database of system representations and offered its users a complete picture of the state of the subsystem. When a new version of a part of a subsystem was created, this new version always appeared within the context of a version of the complete subsystem. Tools then operated on a complete subsystem without having to understand the organisation of versions of individual components of the subsystems.

Our analysis of the requirements for CM and the above systems led us to the belief that VM and CM on object bases is qualitatively different from VM and CM on file systems. The following sections describe the reasons for this belief.

VM and CM for file systems,

In many VM and CM tools for file-based environments we observe that files are treated essentially as almost independent entities. There are no explicit links between files to represent the semantics of the relations between them. There may be some conventions on naming (for example, .o files are created from .c files etc) but these dependencies are not explicitly represented in the file system.

The range of reference points from which a navigation (a pathname) can start to reach an object of interest is also small - the file system root and the current directory (though mechanisms exist to shorten the writing of the path when it passes through some directories e.g. $HOME).

Files do not have attributes which might be used to charaterise properties of their contents.

There are a number of systems that deal with the efficient storage of these individual entities (RCS [TICH85], SCCS [ROCH75] etc) and most of these systems also impose a naming policy on the identification of the versions managed according to the efficient storage scheme. The above two characteristics, and the fact that navigation to identify a file has so few reference points from which to start, have led some people to believe that version management is, in part, a name completion problem. One simply needs to add a version-specific suffix to object designations in the form of pathnames.

Selecting a configuration in such a file-based context involves selecting a set of "independent" entities. The major problem is ensuring that the set is consistent and several approaches are used. In some extensions to file systems, attributes are used to characterise files and selection of files is checked by consistency rules on attribute values (SHAPE [MAHL88]). Other systems break the selection space down by separating the choices into a choice of an interface representation and a choice of an implementation with automatic management at least of the consistency of files containing interfaces and implementations (ADELE [ESTU85]).

VM and CM for object bases.

In object bases, objects are not independent entities but derive at least part of their semantics from their relationships with other objects. Objects have attributes that can be used to characterise the objects and links that can be used to express dependencies. Navigation to an object can be achieved by starting from any object in the object base known to the navigating process and need not be via a root object or a current directory. There will be many paths to an object, in general.

Representing a single object's versions efficiently is useful in object bases but much less important than the ability to make versions of groups of linked objects (and have these stored efficiently).

Some object bases (for example, PCTE's OMS) provide an ability to navigate to objects along any of the relationships to that object and do not provide access to objects via a unique object identifier. This means that the designation of a particular object of the base can be achieved by several different pathnames. In such a system, it is difficult to view version selection as name completion.

Constructing a configuration on an object base involves selecting a set of subgraphs of the object base and combining them together into some larger subgraph (that we call a configuration) if they are compatible and with as little loss of information deriving from the links leaving the constituent subgraphs as possible. This is critical in cases where tools that will be run on the configuration will navigate from objects in the configuration along the links.

3. The Pact approach to Configuration Management.

The Pact project has built VM and CM facilities on the PCTE interfaces. The approach has been to identify a common service (the Version Management Common Service), providing basic version management capabilities for groups of linked objects, and tool

groups, each supporting particular CM functions. These are described more fully in a separate submission to the workshop [OQUE89]. The remainder of this section outlines the build tool set that is under construction for the OMS.

The Build Tool Set.

The objective of the tool set is to provide at least the facilities that are available in build tools currently running on file systems within the more complicated context of a SEDB in a way that respects the aims of the SEDB.

One of the problems to be resolved is the separation that exists between the textual description of the dependencies and the representation of these as links within the base, as maintained by other tools operating on the base. Another is that one can rebuild attributes, links (or sets of these) as well as objects.

We have decided to separate the descriptions of the goal structure, describing the goals and subgoals to be attained during a build, from the designation of the objects in the base that may be associated with a particular goal. (In *make*, for instance, the goal names may also be the pathnames of files associated with the goals).

Each goal may have an explicit condition and may have an associated action. The goal structure, which is explicitly represented in the object base, can expand during the build to represent the actual goals evaluated during the build.

The association between a build goal and an OMS object can be made in several ways:

(i) by name;

(ii) by a query associated with a goal and using object(s) associated with the supergoal or subgoal(s);

(iii) in the absence of a query, using information stored in the metabase characterising link types by the dependencies that they represent.

The design of the build tool group is currently being carried out. There is, at present, no language available for the description of the goal structure or the associations with objects. Our investigations lead us to believe that an upward compatible extension of the syntax of make would be easy to define.

References

Boudier G., Gallo, F., Minot, R., Thomas, I., "An Overview of PCTE and PCTE+", in Proceedings of the 3rd ACM Symposium on Practical Software Development Environments, Boston, November 1988.

Estublier, J., "A Configuration Manager: the ADELE Database of Programs", in Proceedings of the Workshop on Software Engineering Environments for Programming-in-the-large, Harwichport, June 1985, pp. 140-147.

Leblang, D. B. and McLean, G., "Configuration Management for Large-scale Software Development Efforts", in Proceedings of the Workshop on Software Engineering Environments for Programming in the Large, Harwichport, June 1985, pp. 122-127.

Mahler, A. and Lampen, A., "An Integrated Toolset for Engineering Software Configurations", in Proceedings of the 3rd ACM Symposium on Practical Software Development Environments, Boston, November 1988.

McGuffin, R.W., Ellison, A.E., Tranter, B.R., Westmacott, D.N., "CADES: Software Engineering in practice", in Proceedings of the 4th International Conference on Software Engineering, Munich, Sept 1979. xyz?

Oquendo, F., Berrada, K., Gallo, F., Minot, R., Thomas, I., "Version Mechanisms on the PCTE's Object Management System for supporting Version and Configuration Management Tools", submitted to the Second International Workshop on Software Configuration Management, to be held in Princeton, USA, October 1989.

Rochkind, M. J., "The Source Code Control System", IEEE Transactions on Software Engineering, Vol. 1, No. 4, 1975, pp. 364-370.

Tichy, W. F., "RCS - A System for Version Control", Software - Practice and Experience, Vol. 15, No. 7, 1985, pp. 637-654.