PCTE - SOLVING CASE DATA INTEGATION

Kari Rossi
Kim Portman

Software Engineering Department
Nokia Research Center
P.O.BOX 156
SF-02101 Espoo
Finland

PCTE is a programming interface which provides basic operating system primitives for the developer of an integrated software engineering environment. PCTE is designed to be upward compatible with the UNIX[1] operating system. It was defined in an ESPRIT project during 1983-1986.

PCTE replaces the file system of traditional operating systems by an object management system, which serves as a base for data integration. The data model of the object management system is a derivation of the Entity-Relationship model. PCTE is aimed to be the kernel for building software engineering environments for 1990's.

Keywords: CASE, Object Management Systems, PCTE, Software Engineering Environments, UNIX.

[1]UNIX is a trademark of AT&T.

1. INTRODUCTION

There is a large amount of data to be managed in a software engineering environment (SEE). This data is highly structured and the different data elements are closely related to each other. The software development data consists of information like:

- all forms of programs; sources, objects and executables
- documents; requirements, specifications and designs
- project management data; plans, schedules and resources
- relations between different development data
- multiple versions of software development objects.

Current operating systems are not well equipped to manage all this data in an integrated and efficient way. The traditional character oriented file systems do not provide sufficient means to structure the design data. PCTE is a programming interface designed to support particularly the management of software engineering data.

PCTE was defined during 1983-1986 in an ESPRIT project, participated by Bull, GEC, ICL, Nixdorf, Olivetti and Siemens /PCTE87/. PCTE interface definition was placed in public domain. Full implementations are already available for several different workstations. PCTE is gradually gaining acceptance among software tool writers. It is currently used in several European research projects developing tools for SEEs.

This paper describes PCTE basic functionalities and analyses it by comparing it to UNIX. The paper is based on the material in references /Bach86/, /PCTE86a/, /PCTE86b/, /Boud88/, /Camp87/, /Camp88/, /Lyon87/, /Thom88/ and /Port88/.

2. NEED FOR A TOOL ENVIRONMENT

Traditionally software has been developed in centralized computer systems using separate text oriented tools. Recent developments in computer technology have made personal workstations available for software engineers. These workstations are characterized by high processing power, high resolution graphics and operation over local area network. This hardware base is a required step in achieving the long waited productivity gains in software development.

Although these achievements the software base to take advantage of the current hardware is still largely unavailable. The effort for developing software engineering tools for highly integrated environments is much greater than for traditional environments. These new tools are expected to provide increasing levels of services and support for the software developer. The tools are required to be graphics oriented, utilize database technology and support operation in a distributed environment.

The services of traditional operating systems are insufficient for building these sophisticated software engineering tools. These operating systems, including UNIX, were designed to process mainly character oriented information, store data in untyped files and support the work of individuals. To tap the power of workstation technology, better operating environments are needed for building future engineering tools.

PCTE provides a Public Tool Interface (PTI) for tool developers. The interface definition is non-proprietary and it contains services valuable for creating highly integrated tool environments. PCTE addresses some of the key requirements of tool developers:

- it contains graphics services for user interface construction
- it provides data storage in an object management system
- it supports operation in a distributed and heterogeneous workstation environment.

3. PCTE BASIC CONCEPTS

3.1 Processes

PCTE process management is based on X/OPEN-standard /XOPE87/. Most of the X/OPEN calls are included as such, but some calls have been added and some extended. Programs are run in UNIX processes. Each process has a system wide unique identifier so it is possible to refer it even from other workstations.

UNIX system calls are extended in the process invocation. Fork- and exec -primitives are not required from an implementation, they are replaced by call and start primitives:

- call creates a new process which starts to execute the specified program, and the original process waits for the termination of the created process
- start is similar, except that the original process doesn't wait for the termination of the created process.

Processes communicate via pipes, signals and message queues. Pipes and signals are UNIX compatible. PCTE message queues are an extension of UNIX message queues. They are named so that unrelated processes can also exchange messages easily. Many processes may write to a message queue at the same time but only one can read it.

One of the key features in PCTE is distribution. Both processes and object management system are distributed. The distribution is as transparent as possible to the user. Therefore it is possible to execute processes and manipulate data in the same way from all connected workstations.

3.2 Object Management

In PCTE the file system of traditional operating systems is replaced by an object management system (OMS). OMS is based on ER-model /Chen76/ where objects have attributes and relations. Like conventional databases OMS also supports transactions.

OMS objects are typed. There is a type hierarchy and therefore subtypes of an object type inherit definitions from its parent type. The root object type is called 'object'. The most important subtype of this type is 'file', which has contents like ordinary files in UNIX.

Type definitions are described in a data definition language (DDL). The following is an example of DDL definitions:

```
source_file: subtype of file ;
C_module: subtype of source_file ;
project: subtype of object ;
```

3.2.1 Attributes

There are four attribute types in OMS:

- integers
- booleans
- strings
- dates.

Attributes can be associated either to objects or to links. They are used to store typed information in the database. Attributes can be accessed and changed by the user and they can have default values on initialization. Below are some examples of attribute type definitions:

```
project_name: string ;
reserved : boolean := true ;
error_code: integer := -1 ;
```

3.2.2 Links and Relations

Objects in OMS are always referred to by links. The root object is referred to by '_'. It is called the 'common_root' since it is common through all workstations. Hence, OMS objects don't actually have names, only links have. In fact, the only way to create an object is to create a link from an existing (origin) object to the new (destination) object.

Objects are referred by their path names. A path name is a sequence of link names separated by '/', e. g., '_/compiler.system/scanner.c'.

OMS links are classified to three categories:

- composition
- reference
- implicit.

An OMS object is created by establishing a composition link from an existing object to it. An object is deleted when the last composition link leading to it is deleted. Reference links are used to refer to existing objects. Each link has a reverse link, and if the category is left unspecified it is implicit.

The cardinality of a link is either one or many. If the cardinality is one an object may be origin to only one link of that type. Otherwise many such links are allowed. In that case the different links are distinguished by a key. The following DDL definitions extend the previous examples:

```
user_name : string ;

works_in: reference link (project_name) to project ;

user: subtype of object
 with
 attribute
  user_name ;
 link
  works_in ;
end ;
```

A relationship is composed of two link types leading from one object to the other and vice versa. The purpose of relationship types is to preserve integrity between two link types.

3.2.3 Schemas

Each type definition belongs to a schema definition set (SDS). For example, the predefined schema 'sys' contains the following definition for the PCTE common root:

```
new_sds sys is
 ...
 common_root : subtype of object ;
 ...
end sys ;
```

By import and extend facilities it is possible to use existing type definitions. For example, in the schema below some type definitions are imported from the 'sys' schema. The definition of 'common_root' is also extended to have a link to C-program development objects:

```
new_sds simple_C_development is

 import sys-object as object ;
 import sys-file as file ;
 import sys-common_root as common_root ;

 C_module : subtype of file ;
 C_header : subtype of file ;

 name : string ;

 c : composition link (name) to C_module ;
 h : composition link (name) to C_header ;
```

```
relationship (
  header : reference link (name) to C_header ;
  module : reference link (name) to C_module
) ;

extend common_root
  with
  link
    c ;
    h ;
  end common_root ;

end simple_C_development ;
```

At a particular time the objects visible to the user depend on the working schema. Working schema is the union of the schemas activated at each time. The types are resolved in the order they are loaded in the working schema.

When the previous schemas 'sys' and 'simple_C_development' are loaded into the working schema, the visible type definitions is the sum of the types in those schemas. The user may refer to C-modules and C-headers the following way:

- to the objects '_/test.c' and '_/defs.h'
- to the object '_/defs.h' by the relation '_/test.c/defs.header' leading from 'test.c' to 'defs.h'
- to the object '_/test.c' by the relation '_/defs.h/test.module' leading from 'defs.h' to 'test.c'.

3.3 User Interface

The most important concept of the PCTE user interface (UI) is the window. The screen may contain several overlapping windows. They are used to display application output and catch user inputs. Current window is the one where the cursor is at a particular moment, and applications receive all input from the current window.

Other important concepts of PCTE UI are frames and viewports. Application programs output through the frame data structure. There are three types of frames:

- bitmap for bit charts
- graphic for primitive graphics, e.g. circles and lines
- text for textual output.

Before an application may write to a frame it must be connected to a window. The connection is established via a viewport data structure which defines which part of the frame will be visible. In addition, PCTE UI provides primitives for manipulating icons, cursor, carets (position within a window) and fonts.

4. ANALYSIS OF PCTE INTERFACE

4.1 Process Management

PCTE process management is highly compatible with UNIX. However, there are some slight advantages in the PCTE primitives.

One advantage is gained by using call and start primitives instead of UNIX fork- and exec- calls. The most common way of using the UNIX calls is to call exec as the first statement in the child process right after fork is called. Fork starts a new child process which executes the same code as the parent process. This is done by duplicating the parent process memory images to the child process. There is clearly an unnecessary overhead when the new process as its first operation changes its memory contents again.

Another benefit of PCTE is the distribution of processes. PCTE primitives allow the processes to be created transparently in other workstations. For example, when the user types the command

compile main_program.a

in the command interpreter, the first step in the execution is to find out in which workstation the program 'compile' is located - if it is not in the workstation of the user 'compile' is run remotely. Therefore PCTE processes are referenced by network wide process identifiers. Also PCTE message queues offer slight advantages, since the communicating processes can refer to a message queue by its name instead of its queue identifier.

As a summary, PCTE provides some but not tremendous advantages in process management when compared to UNIX.

4.2 Object Management

In the research community there is a common understanding that traditional databases and file systems are not sufficient as software engineering data repositories. Since this is the key component in achieving data integration the environment must support sophisticated manipulation of software engineering objects. The most important requirements for software engineering data management are /Bern87/:

- efficient storage of large amount of data
- control of concurrent accesses, consistency and protection
- support for multiple versions of data
- support for multiple data representations, interface to different languages, tools and hardware
- support for flexible and powerful operators, e.g., manipulation of syntax trees
- support for variable length objects whose internal structure is hidden from the system, e. g., large text files
- support for flexible data types, complex objects and arbitrary types supported by the programming languages.

When compared to these requirements it is obvious that PCTE OMS is a better platform for software engineering tools than hierarchical file systems. Its ER-model supports modelling of complex data: type hierarchy of objects, attributes, links and relations and grouping of related type definitions into schema definition sets. Furthermore, composition links are a natural way of modelling composite objects, i. e. objects, which consist of other objects. A versioning model has also been developed which provides basic facilities for managing versions of (composite) objects /Thom88/.

The only structuring mechanism in hierarchical file systems is a directory. In fact, directory hierarchy could be modelled easily in OMS with an object type having link types to itself and to file type:

```
new_sds simple_hierarchical_file_system is
  ...
 dir: composition link (name) to directory ;
 f: composition link (name) to file ;

 directory : subtype of object
 with
 link
  dir ;
  f ;
 end ;

end simple_hierarchical_file_system ;
```

Hence, all structures that can be modelled in a hierarchical file system can be modelled in OMS but not vice versa.

Still, OMS is far from the optimal solution. The basic types are predefined and operations can not be associated to types. In addition, although OMS doesn't impose restrictions to the size of objects, it is not practical to model very small granularity objects with it.

4.3 User Interface

When PCTE user interface was defined no widely accepted windowing systems existed. Therefore PCTE contains the UI although there are hardly any software engineering specific needs for windowing systems.

However, currently there is a strong standardization effort going on, e.g., X/Windows, Open Look and OSF/Motif. PCTE user interface could be regarded as yet another X/Windows tool kit.

As a summary, it is not the purpose that PCTE user interface definition would compete with the emerging windowing system standards. This has also been noticed by the PCTE Interface Management Board (PIMB) and hence, the emerging windowing standards will certainly have an influence on PCTE UI.

5. CONCLUSIONS

From the previous comparison it can be seen that PCTE is technically a better platform for software engineering tools than UNIX due to its object management system. However, it is not yet a commercial solution since there are not many examples of using PCTE in complex practical problems. Commercial

implementations are available from one source and only recently it has been ported to other workstations besides Bull SPS7 and SUN 3.

One disadvantage of PCTE is that it is implemented inside the operating system kernel. This makes it difficult to port it to other environments. Also some parts of the current implementation, particularly the user interface, have certain performance problems. Still there are some doubts whether an implementation outside the operating system kernel would be feasible.

The future development of PCTE is done in PACT- and PCTE+ -projects. There is also a technical committee in ECMA studying PCTE standardization. The result of this standardization effort will likely be that PCTE will be more widely accepted. Although the rapid progress, it will be well in the 1990's until PCTE will be found in commercial CASE tools.

GLOSSARY

CASE	Computer Aided Software Engineering
DDL	Data Definition Language
ECMA	European Computer Manufacturers Association
ER	Entity-Relationship
OMS	Object Management System of PCTE
PACT	PCTE Added Common Tools
PCTE	Portable Common Tool Environment
PIMB	PCTE Interface Management Board
PTI	Public Tool Interface
SDS	Schema Definition Set
SEE	Software Engineering Environment
UI	User Interface

REFERENCES

/Bach86/ Bach M. J., The Design of the UNIX Operating System, Prentice-Hall, 1986.

/Bern87/ Bernstein P. A., Database System Support for Software Engineering, An Extended Abstract. 9th International Conference on Software Engineering, March 30-April 2, 1987, Monterey California, USA, pp. 166-178.

/Boud88/ Boudier G. & Gallo F. & Minot R. & Thomas I., An Overview of PCTE and PCTE+. ACM SIGSOFT Software Engineering Notes, Vol 13 (1988), No 5, pp. 248-257.

/Camp87/ Campbell I., Standardization, Availability and Use of PCTE. Information and Software Technology, Vol 29 (1987), No 8, pp. 411-414.

/Camp88/ Campbell I., Emeraude Portable Common Tool Environment. Information and Software Technology, Vol 30 (1988), No 4, pp. 210-217.

/Chen76/ Chen P., The Entity-Relationship Model: Towards a Unified View of Data. ACM Transactions on Database Systems, Vol 1 (1976), No 1, pp. 9-36.

/Lyon87/ Lyons T. & Tedd M., Recent Development in Tool Support Environments: CAIS and PCTE. Ada User, Vol 8 (1987), Supplement, pp. 65-72.

/PCTE86a/ PCTE Functional Specifications. Bull, GEC, ICL, Olivetti, Nixdorf, Siemens, Fourth Edition, 1986.

/PCTE86b/ PCTE Ada Functional Specifications. Bull, GEC, ICL, Olivetti, Nixdorf, Siemens, First Edition, 1986.

/PCTE87/ PCTE: A Basis for a Portable Common Tool Environment, Project Report. Esprit '86 Results and Achievements, Elsevier Science Publishers B.V., 1987, pp. 53-71.

/Port88/ Portman K. & Rossi K., PCTE - CASE Vendor's UNIX (In Finnish). FUUG UNIX -88, Finnish UNIX User's Group Meeting, November 16-17, 1988, Espoo, Finland, pp. 7-14.

/Thom88/ Thomas I., Writing Tools for PCTE and PACT: The How-to-do-it Guide. ESPRIT '88 Putting the Results Together, Elsevier Science Publishers B.V., 1988, pp. 453-459.

/XOPE87/ X/OPEN Portability Guide (January 1987), Vol. 1-5. Elsevier Science Publishers B.V., 1987.