An Overview of PCTE and PCTE+.

Gerard Boudier, Ferdinando Gallo, Regis Minot, Ian Thomas*

G.I.E. Emeraude, Bull, PC 58F25, 68 Route de Versailles, 78430 Louveciennes, France.

ABSTRACT

The PCTE project has defined a Public Tool Interface on which Software Engineering Environments can be constructed. The interface definition was put into the public domain in September 1986 and several implementations on several machines now exist. The PCTE+ project was set up to define a Public Tool Interface, based on the PCTE work, that could also serve for the development of defence and other high-security applications. This paper summarises the current status of PCTE activity, presents the principal concepts of PCTE and the evolutions that are being proposed in the PCTE+ project.

1. Introduction.

The PCTE (Portable Common Tool Environment) interface specification defines a Public Tool Interface to be used as the basis for the construction of Software Engineering Environments (SEEs). The construction of a SEE containing a rich set of tools is an expensive undertaking requiring considerable effort. Since this effort is beyond the means of most organisations, it is desirable to find a way of enabling different tool producers and vendors to contribute tools to the environment. One part of the solution to this problem is the definition of a Public Tool Interface. The interface is Public to make it independent of a particular vendor and hardware, a Tool Interface as it provides all of the facilities required by writers of software tools.

The PCTE (Portable Common Tool Environment) interfaces were developed in the PCTE project which was a pan-European project to specify, design and prototype a Public Tool Interface. The project began in October 1983, completing its first phase in September 1986 with the publication of the C language definitions of the interface. The interface is programmatic, defined as a set of primitives that may be called by programmers of tools. The second phase of the project, whose topics included the examination of a number of proposed evolutions of the interface and the adaptation of the user interface definition in the light of the growing importance of X-Windows [XWIN87], ended in June 1988.

The Current Status of PCTE activity.

The C language specifications were placed in the public domain in September 1986 [PCTE86]. They are in two volumes, Volume 1 covers the Basic Mechanisms and Volume 2 the User Interface Primitives. Ada language specifications for Volume 1 became available in May 1987. Work is in progress to define the Ada specifications for Volume 2 and this work will take into account the proposals for interface evolution developed in the second phase of the PCTE project.

Change requests to the interface definition are managed by the PCTE Interface Management Board (PIMB), a multinational committee consisting of representatives of the original partners of the PCTE project, other interested companies, representatives of the software industries of all of the member countries of the European Community, computer manufacturers as well as NATO and the European Space Agency. A Technical Committee (TC33) of the European Computer Manufacturers Association (ECMA) is currently examining the interface definition with a view to European and subsequent international standardisation.

At least two implementations of the interface definition are currently running. One was developed by Olivetti within the PCTE project, as part of the prototyping activity. The other, industrial-quality, implementation was developed in parallel with the interface definition by the Emeraude consortium [CAMP88]. The Ada bindings for Volume 1 have been implemented on the Emeraude implementation and an Ada compiler has been validated on Emeraude running on Sun 3 and Bull SPS7 machines.

Unix and System V are trademarks of AT&T Bell Laboratories.

* Author's present address: Intecs, Via Fratti 4, 56100 Pisa, Italy.

More information on some of the activities surrounding PCTE is given in [THOM88].

The Objectives for the PCTE Interface Definition.

The PCTE interface definition had a number of goals and design constraints. Some of these were:

(i) the definition of a "complete" interface, sufficient for all of the needs of tool writers.

(ii) support for tools written in a variety of languages by the provision of bindings for several languages.

(iii) a well-defined migration path for existing tools. This has been achieved by ensuring that the PCTE primitives are upwardly compatible with Unix System V primitives in the X/OPEN standard. The Emeraude implementation of the PCTE interfaces will run Unix System V executable programs without change. For example, it is possible to start a transaction, run the unchanged executable of a standard Unix editor such as vi on an object of type file or one of its subtypes, and then abort the transaction with roll-back effects on the edited objects of the object base.

(iv) rapid availability of implementations of the interface.

The Contents of the PCTE Interface Definition.

The PCTE Interface Definition is divided into two volumes. Volume 1 contains a description of the Basic Mechanisms, Volume 2 contains a description of the User Interface primitives.

Volume 1 (Basic Mechanisms) is divided into six chapters:

* OMS (Object Management System); the OMS is the data repository of PCTE. This chapter also contains descriptions of the schema management and a proposal for a Data Definition Language (DDL).
* EXE (Execution); the facilities provided for process management.
* IPC (Inter-Process Communication); the facilities provided for exchange of data between processes.
* ACT (Activities); the concurrency control and integrity management facilities.
* COM (Communication); the facilities equivalent to classical file input/output.
* DIS (Distribution); PCTE provides transparent distribution of the data management and process execution facilities. This chapter describes the facilities necessary for the configuration and management of the distributed PCTE environment.

Volume 2 (User Interface) is divided into a number of chapters, with the majority of chapters corresponding to entities manipulable by the tool writer.

These chapters deal with cursors, fonts, icons, menus, screens, scrollbars, viewports and windows.

Additional chapters describe frames. A frame is a data structure into which a tool can write graphical or text information. The frame records the logical description and presentation of the tool's information. The PCTE UI provides text and graphic frames as well as general frame management operations.

Input is dealt with in the selection and input chapters. Finally, end-user characteristics are discussed in chapters on the user and the end-user image.

PCTE+: The Definition Phase.

IEPG (Independent European Programme Group) is a grouping of European members of NATO. In 1987, aware of the growing importance of PCTE and the possible benefits of its dissemination and use amongst its members, IEPG decided to finance the PCTE+ project to examine how to build on the experience of the PCTE interface definition to define the basis for environments suitable for miltary as well as civil applications. The project is divided into two phases; a definition phase and an evaluation phase. The definition phase began in 1987 and will last until 1990.

The terms of reference of the definition phase specified a number of areas where interface evolutions should be studied:

- security
- Unix (and other operating system) independence

but particularly required that the resulting proposed standard tool interface be multi-language and suitable for civil and defence purposes.

The project began by producing a requirements document for the interface definition (the EURAC [EURA87]) which was a revision of the RAC for CAIS-A [RAC86]. The EURAC took account of a number of requirements whose importance had been established by projects building environments on the PCTE interfaces (e.g. Pact [THOM88] and Eclipse [CART87]) in addition to other proposals.

There are obvious advantages to both communities of a common standard for civil and defence work. There is close contact between the PCTE+ designers and the technical group that advises the PCTE Interface Management Board on proposed modifications to the PCTE interfaces. The PCTE+ project has already decided to align itself with the PCTE interfaces for the user interface part of its interface definition.

Both C and Ada specifications are being developed simultaneously within the project. Two editions of the proposed interface definition have been produced and subjected to international review and comment [PCTE88]. The final edition is scheduled to appear in October 1988 to be used as the basis for the PCTE+ evaluation phase.

Structure of this paper.

For reasons of space, this paper does not describe the User Interface primitives of PCTE/PCTE+. Further details of these, including the complete specifications, may be obtained from the authors.

Chapter 2 of this paper is divided into sections following the division of Volume 1 of the PCTE C interface specifications. Each of the sections first describes the features of PCTE, then the additions/changes proposed in the PCTE+ interface definition with a brief rationale for the changes. Three new sections are added that are not present in the PCTE specifications: SEC (Security), NTF (Notify) and ACC (Accounting).

The paper ends with some brief conclusions about the PCTE interfaces.

2. The PCTE and PCTE+ Basic Mechanisms Interface Definitions.

This chapter presents the features of the PCTE mechanisms followed by a brief presentation of the principal PCTE+ proposals for changes/extensions.

2.1. OMS (Object Management System).

The Object Management System is the information repository of PCTE. It is based on the Binary Entity-Relationship model [CHEN76]. The object base is transparently distributed over a local area network.

This section describes the principal characteristics of PCTE's OMS. We describe objects, links and relationships, and attributes. The schema management possibilities of PCTE are then described and this is followed by a discussion of the proposed PCTE+ extensions to the OMS.

2.1.1. Objects, Links, Relationships and Attributes.

This section describes the basic characteristics of the OMS data model. A discussion of how this model meets the requirements for data repositories in SEEs can be found in [GALL86].

2.1.1.1. Object Types and Objects.

Objects in the OMS are typed. An object type is defined by a name (which is not globally unique - see the later section on schemas), a parent type, a set of attribute types, a set of link types for which the object type may serve as an origin and a set of link types for which the object type may serve as a destination.

Object types form a type hierarchy with a subtype inheriting the attributes and links that are defined for its ancestor types. The root of the hierarchy is a predefined type called "object" with a number of predefined attributes. There are several predefined subtypes of the object type. Some of these, for example, file, pipe and message queue have a special predefined property called the contents. The internal structure of the contents is not managed by the OMS and the semantics of some operations performed on an object with contents depend on the type of the object. For example, the Unix operation **open** performed on an object of type file (or one of its subtypes) opens its contents.

The contents is inherited by subtypes in the normal way. Only an object type that has one of the file, pipe or message queue object types as an ancestor type has contents. The OMS therefore generalises the notion of a file system - the Unix file system can be emulated as a specialisation of the OMS but the OMS offers much richer data modelling facilities than those provided by a file system. The unstructured byte stream that is provided by file systems is only a small part of the data modelling possibilities of the OMS.

2.1.1.2. Relationship and Link Types and their instances.

Relationships can be established between objects. A relationship is a bi-directional association between two objects; it can also be seen as a pair of mutually inverse links.

Links are typed. A link type is defined mainly by a name (again, not globally unique), a cardinality defining whether one or many links of this type may start from the same origin object, a set of origin object types, a set of destination object types, a set of attribute types some of which may play the role of key attributes if the link type has cardinality many.

A relationship type is a pair of link types, its properties being defined by the properties of each of them.

There are two other properties of a link type that are significant for the discussion of PCTE+ extensions. Each link type has a category and may also have the stability property.

Link Type Category.

PCTE defines three categories of link: composition, reference and implicit.

Composition links (or, more precisely, instances of link types that have the category composition) have semantics that are closely related to the existence of objects. An object is created with a composition link pointing to it and ceases to exist when the last composition link to it is deleted (a precondition for the success of the operation being the absence of any other link to the object). There can be more than one composition link leading to an object - the graph of composition links can thus form a DAG (Directed Acyclic Graph) or may even contain cycles. The graph of composition links in the object base is certainly not constrained to be a hierarchy.

Reference links are used to represent the many associations that exist between objects in the data repository of a SEE. They do not have the same existence semantics as composition links except that the existence of a reference link to an object will prevent the deletion of the last composition link to it and thus its deletion from the base (as mentioned above). The rationale for this is that one should not be able to delete an object from the base if some other object is referring to it.

Implicit links have limited expressiveness. They may not have any attributes. If the implicit link type has cardinality many then an integer key is generated by the PCTE system.

All PCTE associations are bi-directional, each is a pair of mutually inverse links. A relationship defines a bi-directional association between two objects and is used when the data modeller wishes a certain facility of navigation from either one of the objects to the other. To model a functionally one-directional connection between two objects, since PCTE only has bi-directional associations, a link type is used in one direction and an unnamed implicit link type (the system reverse link) in the other.

Link Type Stability.

The second property of link types that is relevant for the discussion of PCTE+ extensions is the stability property. When a stable link, i.e. one whose type has this property, is created to an object, the object becomes stable and may not be written to until the link is removed. The stable link effectively prevents the modification of attribute values, the contents if the object has contents, and the creation of links (except implicit links) leaving the object.

2.1.1.3. Attribute Types and Attributes.

Attribute types are defined by a name (again not globally unique) and an initial value. The permissible value types are integer, date, boolean and string.

2.1.2. Schema Management.

The OMS has a schema mechanism. Schema information can be divided into three classes:

- Object, attribute and link type definitions;
- Information on which attributes are "applied" to which object types and which link types.
- Information on the source and destination object types for link types, that is, to which object types the link types are applied.

We will refer to the two last classes of information as application information. There is no need to provide this information for relationship types as it is deduced from the the link types that form the relationship type.

The above schema information is not centralised in a global schema definition for the whole environment. It is distributed over Schema Definition Sets (SDSs), each of which contains a subset of the schema information. The SDSs do not partition, in the mathematical sense, the schema information. For example, application information for a single object type can appear in several SDSs. A SDS is a predefined subtype of the file object type.

The names of types are local to a SDS. The same name may therefore be used in several SDSs without name clashing.

Each process has a working schema that defines its visibility of the object base. The working schema defines the types of object, link and attribute that can be accessed and also defines which applications of attributes to objects and to links, and which applications of links to objects are visible.

A working schema is a well-formed union of SDSs, created for a process by the **set_schema** primitive. A process may also change its working schema by calling the **set_schema** primitive. The **set_schema** primitive defines the visibility for names in the SDSs to be formed into the working schema.

Since it is rare for the definitions of types of objects, or other schema information, found in a SEE to remain static, PCTE provides a mechanism for the evolution of SDSs without "dump and reload" of the object base. For example, if a new attribute is applied to an existing object type definition, all instances of the object type, both existing and newly created, will have a value for the attribute.

2.1.3. The PCTE+ proposals.

A number of extensions have been proposed to the OMS definition in the PCTE+ project. They include: composite entities, version support based on the composite entity proposals, a self-referential model, multiple rather than simple type inheritance, secondary links that do not enforce referential integrity, usage modes for type definitions in a SDS, new properties for composition link types and two new value types for attributes.

2.1.3.1. Composite Entities.

There are occasions in the design of a data model for a SEE, or during manipulation of the objects in a SEE object base, when one wishes to treat a number of individual objects as one whole. For example, a document's logical structure may be represented as different objects in the OMS with links relating them. One might wish to move the entire document from one volume to another (or make a new version of the entire document - see below).

PCTE+ defines the notion of a composite entity, new predefined attributes on objects that are particularly useful when the object is the root component of a composite entity, and primitives that apply to composite entities.

In PCTE+, a composite entity, designated by X, consists of:

- an object X, called the root component of the composite entity;
- the set of objects which are in the transitive closure of the composition links starting from X, called its components;

- the links starting from X and the set of objects defined above.

The semantic of the composition category has been extended to include **is_part_of** as well as the **exists** semantic of PCTE.

New predefined attributes are used to indicate the latest access, modification and change dates of a composite entity, defined as the latest access, modification or change date of one of its components.

New operations that manipulate composite entities include: **composite_list_links**, **composite_delete**, **composite_copy**, **composite_move**.

2.1.3.2. Version Support.

The PCTE OMS provides general data modelling facilities. These may be used to model particular version schemas. For example, an environment designer might choose to define a single object in the base to represent all versions of a system. (Such an object acts as a "directory" of versions, in some sense). This object is then linked to each version via links whose keys are used to distinguish between versions (a number to give version number, a string key to give the name of a variant etc).

The PCTE+ version management support uses the idea of composite entities. New interface primitives (**revise** and **snapshot**) are added to allow the creation of new versions of composite entities (with a single object of the object base as a simple, special case) from an origin version.

Relations between versions are expressed by special **predecessor** <-> **successor** relationships, managed by the system. These are created between all of the component objects of the new version and the equivalent component objects of the origin version by the interface primitives. These relations therefore constitute the version graph. The **predecessor** <-> **successor** relationship guarantees that the origin version is stable, that is, the **predecessor** link has the stability property. The consequence of this is that the origin version cannot subsequently be written to, though it is still possible to create new versions from it. The version graph is not constrained to be a tree. A facility is provided to add predecessor links so that the version graph can represent merging of versions in the form of a directed acyclic graph (DAG).

Some of the links that enter or leave a composite entity "track" the versioning, the precise behaviour being defined by the category and cardinality of the links.

To illustrate the effect of these mechanisms, consider a document organised as a composite entity. One of its chapters refers to a separate software specification document (using a reference link inversed by an implicit link of cardinality many). If we create a new version of the document, we would like the new version also to refer, in the same way, to the software specification document. It is not because we have created a new, identical version of a document that it ceases to refer to the software specification. The relationship between the document and the software specification is maintained over the revision primitive.

2.1.3.3. A self-referential model - the metabase.

Types in PCTE are not represented as objects in the object base. PCTE+ proposes a change in the definition of the SDS management primitives so that they manage a metabase structure in which type and application information is represented. This metabase structure would supplement but not replace the existing SDS objects.

Such a metabase structure has a number of advantages. Firstly, it makes the querying and interrogation of schema information uniform with the querying of other parts of the object base - a single mechanism can be used for both. Secondly, it provides a convenient structure for the association of additional information on the schemas, such as documentation, user interface menus associated with object types, libraries of access routines for the manipulation of objects of a certain type, etc.

2.1.3.4. Multiple Inheritance for Object Types.

Object types in PCTE are organised into a hierarchy. In PCTE+ this is generalised into a DAG (Directed Acyclic Graph) of types (multiple inheritance). An object type may therefore have more than one parent and inherit attributes and link/relationships from all of its parents. In some systems this leads to naming conflicts where two ancestors have the same name for different attributes, for example. The SDS mechanism and, in particular, its name scope rules, ensure that this problem does not arise for PCTE+.

2.1.3.5. Secondary Links.

In PCTE+, processes are represented as objects (see Section 2.2.2.1) as are activities (see Section 2.4.2). Several of the PCTE concepts, such as the locks that an activity has on objects, are now represented as links between these objects and other objects in the object base. These links do not have the same characteristics as links of the three categories (composition, reference and implicit) defined in PCTE. This observation led to the definition of a new category for link types, the secondary category.

The properties of the secondary category of link type are:

(i) it is a truly uni-directional association between two objects. (Section 2.1.1.2 discussed links and relationships and explained that in PCTE all associations are bi-directional).

(ii) referential integrity is not enforced. Deletion of the destination object of a link of a type with the category secondary is not affected by the existence of the secondary link. The link remains in existence but cannot be navigated. The link is deleted when its origin is deleted.

(iii) it may have cardinality one or many, and may have attributes.

Several PCTE concepts are now modelled differently, using secondary links. A PCTE process has a number of reference objects that designate objects in the object base. The designation of these objects in PCTE+ is modelled by the presence of secondary links (of type **referenced_object**) from the object representing the process to the objects in the object base, the name of the reference object in PCTE being the key of the secondary link.

2.1.3.6. Usage modes.

The SDS mechanism in PCTE can be used to control the visibility of type definitions and application information. This is achieved by using the fact that SDSs are objects in the object base and therefore subject to access control mechanisms. The drawback with this approach is that, while it controls visibility of a type, there is no way of specifying the operations that processes that have visibility of the type may perform on instances of the type. It is not possible to make a type available to a process for reading only, for example.

PCTE+ has proposed an extension to the SDS mechanism in this direction. For each type in the SDS, one can define a **usage_mode** that controls the permitted operations on the type by processes that incorporate the SDS into their working schema, an **export_mode** that controls the transmission of rights when the type definition is imported into another SDS, and a **maximum_usage_mode** that is used to limit the range of changes that can be made to **usage_mode** and **export_mode**.

The usage mode mechanism can be used in conjunction with the discretionary access control mechanisms of PCTE+ to provide an "object-oriented" interface to types (where "object-oriented" in this case means associating a set of operations with a type). The discretionary access control mechanisms allow access to a SDS to be limited to certain programs. Putting a type definition in SDSs with different **exploit_schema** access rights and different usage modes in each allows, for example, some programs to have write or create access via one SDS while other programs using another SDS have read access.

2.1.3.7. Exclusiveness for Composition Links.

In PCTE+, a new optional property, called exclusiveness, has also been defined for link types with category composition. An object may only have one composition link pointing to it if this link has the exclusiveness property. It can be used to enforce a tree structure on the graph of composition links.

2.1.3.8. Additional Attribute Types.

Two new value types for attributes have been added to the PCTE set of integer, boolean, string and date. They are real numbers and enumeration types.

2.2. EXE (Execution).

The execution primitives of PCTE provide facilities to create and subsequently manage processes, representing the execution of programs.

2.2.1. PCTE Execution Management.

There is a predefined type of object called a static context that is used to model a program. Programs are therefore modelled as objects in the object base.

A static context may be directly executable, in which case it may have an associated execution class. The execution class is an object linked to some of the objects in the object base that represent stations known to the environment (see 2.5 below on Distribution). The execution class is used to model the subset of stations in the environment on which the executable can execute.

A static context may also be interpretable, for example, a script to be interpreted by a command line interpreter. In this case, the static context is linked to another static context that is its interpreter.

Primitives are provided to start the execution (both synchronously and asynchronously) of static contexts as processes, to terminate (normally and abnormally) a process, and to interrupt (suspend and resume) a process.

2.2.2. The PCTE+ proposals.

There has been a major change in the presentation of these facilities in the PCTE+ proposals. In PCTE, processes are not considered within the object model of the OMS. In PCTE+, there is a unification of the descriptive model for processes and the OMS.

Execution is also a difficult area for operating system independence so PCTE+ makes proposals in the area of foreign systems and processes to deal with the problems of use of existing tools in PCTE+ environments.

2.2.2.1. Processes as objects.

In PCTE+, processes and activities (see Section 2.4) are modelled as objects in the object base. Properties of the process can then be modelled as attributes of the process object or links emanating from or leading to the process objects. The advantage of this approach is two-fold: first, a single model is used for the explanation of the concepts, and secondly, access to the information about a process can use the same model and mechanisms as access to the information about any other object in the object base.

A PCTE+ process is characterised by a number of properties that together constitute its dynamic context. When a process starts another process, some of these properties are inherited by the called process from the calling process, some of the properties are not inherited but are initialised when the object that represents the called process is created and some whose inheritance can be controlled by the setting of attributes on the properties of the calling process. Examples of some of the properties of a process, with an indication of whether the property is automatically inherited, not inherited or whether the inheritance is user-defined are:

- current state of the process - not inherited.
- mandatory security context of the process - automatically inherited.
- set of reference objects currently defined - user-defined inheritance.

2.2.2.2. Foreign Systems and Foreign Processes.

One of the objectives of the PCTE project was to ensure an easy migration path for existing tools running on Unix System V. The interface definition therefore contains some primitives for which the principal motive for inclusion is compatibility with Unix System V.

Since PCTE+ has the express objective of operating system (and hence Unix) independence, we are immediately faced with the problem of the use of existing tools and environments written on interfaces other than PCTE+, including proprietary operating system interfaces. The solution that has been adopted in PCTE+ is to define the notions of a foreign system and a foreign process.

A foreign system is a system providing interfaces for program execution other than those provided by PCTE or a PCTE station belonging to another PCTE environment. Examples of foreign systems might include a bare target machine, a host system connected to the network but running only a proprietary operating system, a PCTE host running an implementation of the PCTE interfaces on top of an operating system interface that still allows other programs to run using the underlying operating system interface etc. Foreign systems are modelled as objects in the object base.

Four types of facility are provided in PCTE+ to support connection to foreign systems.

- modelling of foreign execution sites and static contexts for execution on foreign sites;
- routines to transfer data from between the contents of PCTE objects of type file (or one of its subtypes) and foreign files in the foreign environment;
- the possibility of mapping between PCTE objects of type message_queue (and possibly pipe) to appropriate facilities in the foreign system. Such a mechanism could be used to create a communication channnel between PCTE+ tools and those running on the foreign environment;
- a common monitoring protocol for debugger communication with a foreign target.

2.3. IPC (Inter-Process Communication).

An integrated SEE will be characterised by a high degree of data exchange between the tools that constitute it. The Public Tool Interface must therefore provide mechanisms that facilitate this communication in an efficient way even in the context of a distributed environment.

2.3.1. The PCTE Inter-process Communication Mechanisms.

Three mechanisms are provided in PCTE for inter-process communication: signals, pipes and messages. The first two of these are essential for compatibility with Unix. The third is defined in a similar way to the System V Interface Definition Kernel Extension's definition of messages.

A message queue is used to send and receive messages, a message being an arbitrary block of data, characterised by a user-defined message type. Message queues have identifiers that are system-wide unique so that messages can be sent to message queues on any connected station. Message queues can also be named objects in the object base. This facility can be used to avoid two processes that wish to communicate via a message queue needing prior knowledge of the message queue identifier.

Send operations on a message queue are normally non-blocking but facilities are provided that can be used to increase the level of synchronisation of the communicating processes. A rich set of message queue interrogation and selection facilities is provided for receiving processes.

2.3.2. The PCTE+ proposals.

Inter-process communication is another area that is highly operating system specific. PCTE+ has consequently removed the management of signals from the interface definition. Pipes remain as a useful means of transfer of information and coordination between processes that were written without the knowledge that they were to cooperate in this way.

The relationship between a process and the message queues it has reserved is represented by a link (of type **reserved_message_queue** and category secondary) from the process to the message queue.

2.4. ACT (Activities).

A Public Tool Interface must provide facilities to support concurrent working and to guarantee the integrity of the information in the data repository. PCTE introduces the concept of an activity to meet these requirements.

2.4.1. PCTE Activity Management.

An activity is a framework in which a set of related operations takes place. It may be in one of three classes:

unprotected activities are those that do not require their data accesses to be protected from other concurrent activities;

protected activities require protection from concurrent access for the data they access but do not require atomicity of effect of data modifications;

transactions are protected activities whose effects on the database are atomic.

Each PCTE process executes in the context of an activity. A process may only have one activity that it has started in progress at any one time.

The execution mechanisms of PCTE allow tools to be built by composing individual elementary tools. A child process may, of course, start its own activity (with the restriction stated above) and this allows the possibility of nested activities (and therefore nested transactions).

OMS activities feature a recovery mechanism to support transactions and a locking mechanism that is normally implicit (automatic according to the object base accesses made by a process) but can be used explicitly in some situations, for example, to prevent some of the modifications made during a transaction being rolled back in the event of a transaction abort.

Clearly, all of the above mechanisms work transparently in the normal case of a distributed object base and execution of different processes on different machines of the network.

2.4.2. The PCTE+ proposals.

PCTE+ has proposed a change to the way that these facilitites are modelled. Activities are now modelled as objects in the object base. An activity may have locks on objects and these locks are modelled as links (of type **lock** and category secondary) from the activity object to the locked object. There is also a link (of type **locked_by** and category secondary) from the locked object to the activities that have a lock on it.

2.5. DIS (Distribution).

One of the original design criteria of the PCTE project was the definition of an interface that could be implemented on a distributed system subject to partitioning, for example a local area network, in such a way that useful work could still be done in each of the partitions (the availability criterion). The distribution of both data and program execution was to be transparent. Both of these objectives have been met.

The object base is partitioned into volumes, each containing some of the objects of the base. There is no difference between the creation or manipulation of links between objects on the same volume and the creation or manipulation of links between objects residing on different volumes. The positioning of objects in volumes, and the positioning of volumes on workstations is transparent to the user unless he/she wishes to find out. Volumes are represented as objects in the object base. There are two stages to the use of volumes, initialisation and mounting.

Workstations are also represented as objects. As for volumes, there are two stages: firstly, the declaration of a station to the PCTE environment and its creation as an object, and secondly, the connection of a station to the environment.

There are certain system objects in the object base that are necessary for the operation of the basic mechanisms. Examples of these objects include the system SDS sys, the directories of volumes and of stations, SDSs etc. In order to meet the availability criterion, these objects must be available even when the network is partitioned. This is achieved by replicating these objects. PCTE provides mechanisms for managing replicated objects including, but not limited to, the system objects, and mechanisms to manage the deferred update of copies of replicated objects.

2.6. SEC (Security).

PCTE's access control mechanism resembles that of Unix. The user, group, world paradigm is used.

PCTE+ makes significant proposals in the area of access control and security. It adopts a distinction between discretionary and mandatory access control [DOD83].

Discretionary access control is defined as "A means of restricting access to objects based on the identity of subjects and/or groups to which they belong. The controls are discretionary in the sense that a subject with a certain permission is capable of passing that permission on to any other subject."

Mandatory access control is defined as " A means of restricting access to objects based on the sensitivity of the information contained in the objects and the formal authorisation of subjects to access information of such sensitivity."

PCTE+ supports both discretionary and mandatory access controls and auditing, though auditing is not described here for space reasons.

2.6.1. Discretionary Access Control.

Discretionary access control in PCTE+ operates by comparing the access rights of a process with the access rights necessary to perform the intended operation on objects. A key concept in model is that of the discretionary group.

We first begin by describing the static organisation of the object base data that is used by the discretionary access control. This is followed by a brief description of the dynamic aspects of how this information is used.

Static Information for Discretionary Access Control.

The object base of PCTE+ contains the basic information on which the discretionary access control is based. There are structures representing the organisation of users and programs in the environment, and information on each object.

Users are represented as objects in the object base. In addition to this representation of users, discretionary groups are also represented as objects. There are two types of discretionary groups, discretionary user groups (DUGs) and discretionary program groups (DPGs). Each of the former contains only users (who may belong to more than one DUG). DPGs contain only static contexts (see section

2.2).

DUGs allow a project organisation to be modelled in terms of roles with associated permissions. An example application might be the construction of a Configuration Management policy. DPGs might be used to associate programs with object types as described in section 2.1.3.6 on usage modes.

DUGs and DPGs are organised into lattices, one for DUGs and the other for DPGs. Each arc in the lattices represents the **user_subgroup_of** (or **program_subgroup_of**) relation. Some parts of these lattices are predefined, to reflect that certain PCTE+ interface primitives require special privileges. The supergroups of a discretionary group are those in the transitive closure of the **user_subgroup_of** (or **program_subgroup_of**) relationship.

Each object in the object base has an access control list. Each entry in this list contains the name of a discretionary group (program or user) and the elementary access rights that are granted, denied or undefined for that group on that object. The elementary access rights are : **navigate, read, append, write, execute, exploit_device, exploit_schema, control_discretionary, control_object, control_mandatory.**

Dynamic Aspects of Discretionary Access Control.

Each process has a discretionary context composed of a set of discretionary groups and a set of boolean properties, called its adopt mode, associated with each group. The discretionary context contains one and only one discretionary group of type **user**; one and only one discretionary group of type **user_group**, though when a **user_group** is adopted, all of its supergroups are also adopted; and one and only one discretionary group of type **program_group**, though when a **program_group** is adopted, all of its supergroups are also adopted.

The access rule is simply expressed: a process has a permission on an object if and only if the corresponding access rights are explicitly granted to at least one of the adopted groups in the discretionary context of the process and are not denied to any of the adopted groups of the process.

2.6.2. Mandatory Access Control.

As for discretionary access control, the description of these facilities can be divided into static and dynamic aspects.

Static Information for Mandatory Security Control.

PCTE+ distinguishes between confidentiality and integrity. Confidentiality classes and integrity classes are modelled as objects in the object base, linked to objects representing users cleared to those confidentiality and integrity levels.

Confidentiality classes are linked via a relationship (of type **dominates_in_confidentiality**) that indicates that all users cleared to a class are also cleared to all classes that it dominates. The same applies to integrity.

Each object has confidentiality and integrity labels. The label is a restricted form of expression on the names of the confidentiality (or integrity) classes defined in the environment.

Dynamic Aspects of Mandatory Security Control.

A process is characterised by a confidentiality context and an integrity context. Information flow between objects via a process is such that a process may only:

- read from an object provided that its confidentiality context dominates the confidentiality label of the object;
- write to an object provided that its confidentiality context is dominated by the confidentiality label of the object;
- write to an object provided that its integrity context dominates the integrity label of the object;
- read from an object provided that its integrity context is dominated by the integrity label of the object.

Finally, a process may communicate information to another process if and only if the confidentiality context of the second dominates that of the first and the integrity context of the first dominates that of the second.

2.7. NTF (Notify).

The notify mechanism is an extension of PCTE proposed in PCTE+. To illustrate its utility, consider the following example. In a distributed environment, two workstations are both displaying information on the same object of the object base. If one of the stations modifies the object, it would be useful to notify the other that some change has been made to the object. This notification could then be used to indicate to the user of the second station that the information on the screen is no longer up-to-date. A similar mechanism (the notify lock) has been proposed in [SKAR86].

The notify mechanism permits the designation of a single object, access events, a message queue and notifier messages. Typically a process will connect itself to a message queue to receive the notifier messages, enable a notifier specifying the object to be monitored and the message queue that is to receive the notifier messages, and switch on the access events that are to be monitored by the notifier.

More than one process can monitor an object and a process can monitor more than one object.

As is frequently the case in the design of a Public Tool Interface, a mechanism designed to resolve a particular problem interacts with other facilities of the interface. In the case of the notify mechanism, the most interesting interaction is with the transaction mechanism, particularly in the case of nested transactions. The problem arises in specifying the moment at which the notifier messages are sent to the message queue if the access events occur in a nested transaction (which could be aborted by the roll-back of one of the enclosing transactions). The PCTE+ solution is to say that the message is sent when the locks guaranteeing protection against concurrent access are removed from the object and is sent only to processes within the enclosing transaction.

2.8. ACC (Accounting).

PCTE provides only very limited facilities for accounting for the resource usage of processes in the environment. These facilities are greatly enhanced in PCTE+.

PCTE+ defines the notion of an accountable resource as a selected object in the object base. The objects might be static contexts, pipes, files, stations, SDSs etc. When a PCTE+ process accesses an accountable resource, accounting information is appended to an accounting log object when the access operation terminates. The accounting log objects can subsequently be visited by a person with appropriate privileges (such as an accounting supervisor).

Note that there is also an interaction here with the transaction mechanism. Accounting log information on object base accesses that occur within a transaction is not rolled-back if the transaction is aborted.

3. Conclusions.

The definition of the PCTE interfaces benefitted from the parallel activities of prototyping within the PCTE project and the development of an industrial-quality implementation. Input from the construction of real environments on the PCTE interfaces, and the increased understanding of Public Tool Interface issues mean that PCTE+ has been defined to meet real needs of SEE designers. The first generation of environments using this technology is appearing - it's time to remember that Public Tool Interfaces are only a means to an end. Our real goal is the provision of Integrated Project Support Environments to improve software productivity.

Acknowledgements.

The PCTE and Pact projects were partially funded by the Commission of the European Community's Esprit Research Programme. The partners in PCTE were Bull, GEC, ICL, Nixdorf, Olivetti, and Siemens. The partners in Pact are Bull, Eurosoft, GEC, ICL, Olivetti, Siemens, Syseca and Systems and Management.

The PCTE+ project is being carried out by a consortium of companies: GIE Emeraude, Software Sciences Limited and Selenia. The PCTE+ specifications are principally the work of the PCTE+ Specification Group that includes: Gerard Boudier, Christian Bremeau, John Cartmell, Claudio Costa, Ferdinando Gallo, Jean-Claude Grosselin, Mel Jackson, Tim Lyons, Regis Minot, Flavio Oquendo, Tony Riddiough.

References.

CAMP88 Campbell I., "Emeraude Portable Common Tool Environment", Information and Software Technology, Vol. 30, No. 4, May 1988.

CART87 Cartmell J. and Alderson A., "The Eclipse Two-Tier Database Interface", Proceedings of the 1st European Software Engineering Conference, Strasbourg, Sept 1987.

CHEN76 Chen P.P., "The Entity-Relationship Model: towards a unified view of data", ACM Transactions on Database Systems, Vol 1, No 1, March 1976.

DOD83 Department of Defense Document CSC-STD-001-83, "Department of Defense Trusted Computer System Evaluation Criteria", 15 Aug. 1983.

EURA87 "Requirements and Design Criteria for Tool Support Interfaces (EURAC).", GIE Emeraude, Selenia, Software Sciences Ltd., 17 July 1987.

GALL86 Gallo F., Minot R. and Thomas M.I., "The Object Management System of PCTE as a Software Engineering Database Management System", Proc. Second ACM Symposium on Practical Software Development Environments, ACM Sigplan Notices, V22, No.1, January 1987.

PCTE88 "PCTE+ Ada and C Functional Specifications", Issue 2, July 8, 1988. GIE Emeraude, Selenia, Software Sciences Limited.

PCTE86 "PCTE Functional Specifications 1.4", Bull, GEC, ICL, Olivetti, Nixdorf, Siemens, Sept 1986.

RAC86 "Requirements and Design Criteria for the Common APSE Interface Set (CAIS)", DoD Ada Joint Program Office, 4 Oct. 1986.

SKAR86 Skarra A.H., Zdonik S.B. and Reiss S.P., "An Object Server for an Object-oriented Database System", in Proc. of the IEEE International Workshop on Object-oriented Database Systems, 1986.

THOM88 Thomas M.I. "The PCTE Initiative and the Pact project", to appear in ACM Software Engineering Notes.

XWIN87 XLib - C Language X Interface, Protocol Version 11, M.I.T., June 19, 1987.