The PCTE+'s OMS
A Software Engineering Distributed Database System for supporting
Large-Scale Software Development Environments

Flávio Oquendo
Gérard Boudier
Ferdinando Gallo
Régis Minot
Ian Thomas
G.I.E. EMERAUDE
BULL Corporate Research Center
68, Route de Versailles
F-78430 - Louveciennes
FRANCE

Abstract
This paper presents the PCTE+'s OMS, a software engineering distributed database system. Its main concepts and features are described, including the data model, the view mechanism, the concurrency and integrity control mechanisms, the access control mechanisms, and support for multiple programming languages and database distribution. The paper begins with a review of related work, and explicitly itemizes the requirements for a software engineering database system.

Keywords: Software Engineering Databases, Tool Support Interfaces, Software Engineering Environments, PCTE+.

1. Introduction
Ten years ago, the Stoneman report [Buxton 80] defined an architecture for software engineering environments. This architecture distinguishes clearly between the tools for the users of the environment and a tool support interface to which all the tools are interfaced. The tools provide the functionality to the user of the environment. The tool support interface provides common mechanisms for the implementation and integration of tools. The advantage of building tools using such a tool support interface is twofold: it reduces the amount of code to be produced by tool writers (thus, decreasing the tool cost) and enhances the tool integration (through common mechanisms).

The central mechanism provided by a tool support interface is a database system. This central mechanism is a software engineering database system used to manage all data produced and manipulated by tools, thus factoring out all the services needed by tools for managing their data.

The PCTE+ project [Boudier et al 88] has defined a tool support interface, called "PCTE+", based on the PCTE (Portable Common Tool Environment) interface [Campbell 88], to be used as the basis for the construction of integrated large-scale software engineering environments. This interface is programmatic, defined as a set of primitive operations that may be called by tools. It provides mechanisms for tool execution, tool input/output, tool inter-communication, database management, data integrity and concurrency con-
trol, database distribution, and user interface.

The PCTE+ component that provides distributed database management facilities (including concurrency and integrity control) is the so-called "Object Management System" (OMS). Its objective is to manage all of the objects produced and used in software engineering environments constructed on PCTE+, their properties and their inter-relationships, while enforcing their integrity. The database managed by the PCTE+’s OMS is transparently distributed in a set of workstations interconnected by a local area network. Hence, the PCTE+’s OMS is a software engineering distributed database management system.

This paper focuses on the data model and main features of the PCTE+’s OMS. A review of related work puts the PCTE+’s OMS effort in context. This review is followed by a discussion of the database requirements for software engineering environments. The data model and main features of the PCTE+’s OMS are then described, along with an analysis of how it meets these requirements.

2. Related Work

Several data models and corresponding database systems have been proposed to meet the database requirements for software engineering environments. Much of this early work has influenced our work.

Over the past decade, the issue of database systems for software engineering environments has received considerable attention. The central role that they play in the support of integrated software engineering environments has been recognized since the beginning of the eighties [Buxton 80, [Osterweil 81], [Oberndorf and Penedo 85], [Stenning 87].

The first attempts to address the issue of database systems for software engineering environments were using conventional database systems (i.e. hierarchical, network, and relational).

These attempts (see for instance [Linton 84]) identified several problems, including lack of expressiveness of the data models and poor performance of the database systems. In fact, conventional database systems have primarily been designed for commercial data processing applications. These applications deal with large collections of simply structured entities having an (almost) stable database schema. In contrast, software engineering applications (like other engineering design applications) act on large amounts of complex entities which are highly inter-related. Furthermore, the trial-and-error nature of the design activity leads to a quickly evolving database schema.

Faced with a database application (i.e. software engineering) for which conventional database systems are not appropriate, numerous proposals have been made to overcome the limitations of existing data models and database systems in order to meet the database requirements for software engineering environments.

The major challenges for such a design are: first, designing a data model that is able to describe and manage the entities, their properties and their relationships, produced and manipulated by software engineering environments in a suitable form, and second implementing this data model and other required features in an efficient way.

Three basic alternatives have been exploited:

- to extend an existing data model and the corresponding database system with concepts and features that are appropriate for software engineering environments;
- to develop a new data model for meeting the database requirements for software engineering environments, and implement it using an existing database system (as a back-end);
- to develop a new data model for meeting the database requirements for software engineering environments, and building (from scratch) a new database system to implement it.

An example of the first alternative is IDE’s Software through Pictures [Muller 89]; examples of the second are SKB [Meyer 85], PMDB [Penedo 86], and ALMA [Lamsweerde et al 87].

The third alternative has mainly been exploited by part of the research community that believes that only a specific data model and a special-purpose database system provide the capabilities for satisfying the database requirements for software engineering environments. Furthermore, this database system should be a component of a wider framework for supporting software engineering environments (i.e. tool support interfaces). Examples of this alternative are DAMOKLES [Dittrich et al 86] and the PCTE+’s OMS.

These various data models are often data models defined as specializations of the Entity-Relationship-Attribute (ERA) data model [Chen 76] in order to fit better with the database requirements for software engineering environments. Database systems which are based on these specific data models have been implemented. We now describe three of these cited so far in further detail.

The SKB (Software Knowledge Base) system describes the information of software engineering environments through the concepts of atoms, attributes, binary relations, the constraints that these relations must satisfy, and actions to be taken when the constraints are violated by certain atoms. The SKB data model is based on the binary ERA data model. An SKB database is a "model" of a certain set of software engineering entities and their relationships. The "model" is conceptually and physically distinct from the "real" entities themselves. This approach makes impossible to ensure consistency between the SKB "model" and the "real" information: one cannot prevent users from modifying the entities (for instance, source code of programs stored in a file system) without making the corresponding changes in the "model". This problem is very characteristic of this kind of approach, that is of having the real software engineering entities stored in the file system of the operating system and a database system that modelizes this information. This kind of approach, very common in conventional applications, is unpractical in a design environment such as software engineering. The PCTE+’s OMS overcomes this problem by storing the
real entities (which in reality are “soft”) in the database itself. SRB addresses some forms of semantics constraints in greater depth than we have. In the ALF project [Benali et al 86], [Legait and Oquendo 88] some forms of semantics constraints are being studied [Oquendo 89] in order to enhance the next generation of PCTE+’s OMS.

The PMDB (Project Master Data Base) data model is also based on the ERA data model. The PMDB data model embodies three concepts: objects, attributes, and relationships. Using this data model, the database schema of the information manipulated by the TRW life-cycle methodology was designed [Penedo and Stuckle 85]. This schema comprises 31 objects, approximately 220 attributes, and approximately 170 relationships. [Penedo and Stuckle 85] states that, in the course of this design, at times they encountered problems in representing the information. The PCTE+’s OMS overcomes some of these problems because it includes the concepts of composite entities, attributes applied to either objects and relationships, an union type constructor which allows relationships to be defined on unions of object types, and mechanisms for multiple type inheritance. These features, as observed in [Lamowordec et al 87], can contribute to reduce the complexity of the designed PMDB schema by eliminating semantically redundant relationships and organizing the type structure in a more natural way.

The DAMOKLES data model is called DODM (Design Object Data Model). It is also based on the ERA data model. DAMOKLES, as the PCTE+’s OMS, extends the ERA data model with the notions of composite objects and versions. For supporting uninterpreted data, DAMOKLES provides a new domain for attribute values: “long_field”. Long fields are byte strings of arbitrary length that are manipulated as direct files. DAMOKLES, unlike PCTE+’s OMS, does not provide type inheritance mechanisms, view mechanisms for tools, nor nested transactions for tool composition.

3. Database Requirements for the PCTE+’s OMS

Several works have identified the database requirements for software engineering environments [Bernstein 87], [Dittrich et al 86], [Nestor 86], [Oberndorf and Penedo 85], [Penedo et al 89]. The PCTE+ project itself began by producing a document on the requirements for tool support interfaces, including the database management aspects. This document, known as “EURAC” [EURAC 89], takes into account a number of requirements issued from projects building software engineering environments on the PACT interface, such as PACT [Oquendo et al 89], [Thomas 89].

There is considerable overlap in the requirements which result from these works. Thus, we believe that they are representative of the “real” needs to be satisfied by software engineering database systems.

In this section, we present the twelve main requirements that drove the design of the PCTE+’s OMS.

Requirement 1 - provide facilities for defining entity types, relationship types, and attribute types, as well as facilities for manipulating instances of these types.

A software engineering environment should be able to manipulate many kinds of entities, relationships between entities, and attributes of entities and relationships. Examples of these entities may be projects, tasks, users, documents, requirements specifications, design specifications, program source code, object code, test cases, test data, product releases, bug reports, change requests, and so forth. Examples of relationships between these entities may be “work on” between user and task, “written by” between document and user, “compiled from” between object code and source code, and so on. Attributes hold the actual data about entities and relationships. Examples are the initial date of tasks, the name of users, the date users begin to “work on” tasks, etc. Hence, the PCTE+’s OMS should provide facilities to create, delete, examine, modify, and identify entities and relationships, stabilize entities, examine and modify attributes, and so forth.

Requirement 2 - provide facilities for representing and manipulating data down to the level of fine granularity.

The data that is dealt with in a software engineering environment has a wide range of sizes, ranging from very coarse grains like documents and abstract syntax trees of programs down to fine grains like paragraphs within documents and nodes within abstract syntax trees. Some of these data are in fact composite entities, composed of other entities (possibly recursively), such as documents which are composed of chapters which in turn are composed of sections. Furthermore, the PCTE+’s OMS should provide facilities for storing variable-length chunks of data whose internal structure is hidden from the database system, such as the object code of programs.

Requirement 3 - provide facilities for storing and accessing (in a possible transparent way) multiple versions of single and composite entities while retaining their version history and controlling them.

During the development of a software product, intermediate versions of its components will be produced. Subsequent changes to these components must be strictly controlled since the work of many users can depend on a particular version. Furthermore, the PCTE+’s OMS should provide facilities for supporting configurations.

Requirement 4 - provide facilities for defining new types by extending existing (base) ones while maintaining the compatibility with these base types.

A software engineering environment should support the evolution of the kinds and organization of information that pertain to projects, support the integration of new tools, and accommodate changing project needs. The schema of software engineering databases, as in other design environments, evolves very quickly. Therefore, a software engineering database system should provide facilities for managing rapidly changing database schemas. These changes are mainly the creation of new types and the extension of existing (base) ones. In fact, users need to define new types that are extensions of existing types, for instance for integrating new tools in the environment. A new type must therefore have the same properties as each of the bases, plus additional...
ones. Tools that expect entities of one of the base types will also accept and work correctly on entities of the new type. The tool should not need any kind of recompilation or other preparation to operate on the new type.

**Requirement 5** - provide facilities for defining appropriate database views for tools.

In a software engineering environment, allowing multiple views, and in particular, the guarantee that some definitions will be visible to an executing tool is fundamental in the achievement of portability and integration of tools. In fact, when two tools coming from different sources are installed on the same software engineering environment, there is a high probability that some type definitions will conflict. Views are the ideal way to resolve such conflicts.

**Requirement 6** - provide facilities for allowing dynamic access synchronization mechanisms to entities, relationships, and attributes.

A software project frequently has many users active simultaneously. To allow them to work productively and without mutual interference, it should be possible to isolate the collections of entities they are working with. A user with the access rights to change some particular part of the database needs to be able to “lock out” changes by other users with similar rights, so that he or she can perform a series of operations without interference.

**Requirement 7** - provide facilities for controlling concurrency of tool executions and enforcing the integrity of the database.

Transactions provide a means to support the preservation of the database integrity in the face of concurrently executing tools, and tool and system failure. Locking gives some assistance for preventing clashes due to concurrent programming.

**Requirement 8** - provide facilities for nested transactions.

The transaction mechanism is used by individual tools to ensure the integrity of the data they manipulate. However, tools may be composed together. The use of transactions by the composite tools is also a necessity, implying a need to support nesting of transactions.

**Requirement 9** - provide facilities for specifying and enforcing discretionary access control.

A software engineering database system should provide facilities for allowing the access rights of each user or tool to each entity, relationship, or attribute to be specified. It must also enforce these access controls. Furthermore, it should provide functions to control the propagation of access rights.

**Requirement 10** - provide facilities for supporting tools written in different programming languages (by providing multiple language bindings).

Tools are written in a variety of programming languages. Thus, it is essential that tools can be produced in different languages as well as in a mixture of languages, hence allowing the strengths of particular languages to be exploited in the appropriate circumstances. The PCTE+’s OMS should also permit the interworking of tools written in these different programming languages.

**Requirement 11** - provide facilities for supporting an integrated tool set running on a network of (possibly heterogeneous) workstations and mainframes.

The current trend in the software industry is towards local computing resources, networked to provide project or company computing capacity. Furthermore, modern tools are increasingly “greedy” for processing power and the impact of using such tools on a single mainframe is becoming increasingly unacceptable. Thus, a network of workstations and mainframes is essential for supporting a large-scale software engineering environment.

**Requirement 12** - provide facilities for supporting distribution of the database over the network of workstations and mainframes allowing data to be accessed in a logical way.

Within a database, entities are addressed according to logical relationships with other entities, and not via a physical mechanism.

These twelve requirements highlight the main database requirements for software engineering environments that have direct and important implications for the design of software engineering database systems. Hence, a suitable software engineering database system for supporting large-scale software engineering environments should satisfy at least these requirements, and the PCTE+’s OMS was designed with them in mind. It will be described in detail in the next section.

4. **The PCTE+’s Object Management System**

In the preceding section the main database requirements for the PCTE+’s OMS were presented. In this section we describe the PCTE+’s OMS data model and its main features designed to satisfy these requirements, and discuss how they are satisfied.

4.1 **Concepts of Objects, Links, Relationships, and Attributes**

The first requirement expresses the basic functionality of a database system: to provide mechanisms for defining entity, relationship, and attribute types and for managing their instances. The PCTE+’s OMS satisfies this requirement by providing a specific data model that is derived from the Entity-Relationship-Attribute (ERA) data model [Chen 76] and the object-oriented paradigm by taken into account the database requirements for software engineering environments. Hence, the OMS data model is a software engineering special-purpose data model.

The OMS data model is based on three concepts:

- **objects**
- **attributes**
- **links** (relationships are defined by means of links).

*Objects* are entities (in the ERA sense) which can be uniquely identified. They may have attributes, and be the origin or
Links can be seen as "directed" binary relationships (in the ERA sense). They may have attributes. A subset of the attributes of links, called key attributes, allows the designation of a specific link (among the possibly many of which a given object may be an origin) by their values. Designation of links is the basis for the designation of objects - the principal means for accessing objects in most OMS operations is to navigate the object base by "traversing" a sequence of links.

Relationships are pairs of mutually inverse links such that the origin of each is the destination of the other. Thus, (in the ERA sense) they can be seen as binary relationships.

Attributes have the normal ERA sense. They are used to describe specific properties of objects and links.

Thereby, the OMS database is a set of objects inter-related by links where objects and links are qualified by attributes and some objects have contents.

The objects, attributes, and links are typed. All instances of a given type share a common type identifier and common characteristics (i.e. the type definition). The set of all type definitions describing the structural and semantics properties of the OMS database constitutes the overall schema of the OMS database.

An object type is characterized by:

- a name,
- a set of supertypes,
- a kind of contents (no contents, file, pipe, message queue, or device),
- a set of link types for which it can be either an origin or a destination type.

These attribute types and link types are said to be applied to the object type.

An attribute type is characterized by:

- a name,
- a value type,
- an initial value;
- a duplication property.

An attribute type can be applied to a set of object types and link types. The value types of an attribute type can be integer, float, boolean, date, string, or enumeration. The initial value defined for an attribute type specifies the value of the attributes of this type when an object or link having this attribute is created and no other value is set. The duplication property of an attribute type determines whether or not the attributes of this type are to be copied when the objects or links they qualify are copied. If it is set to false, the attribute is reset to its initial value in the copy.

A link type is characterized by:

- a name,
- a cardinality (one or many),
- a category (existence, composition, reference, designation, or implicit),
- an optional stability property (stable or transitive stable),
- an optional exclusiveness property,
- an optional duplication property,
- a set of possible origin and destination object types,
- a set of attribute types (possibly empty),
- a list of key attribute types (if its cardinality is many).

The attribute types are said to be applied to the link type. Link types in turn can be applied to object types.

The cardinality of a link type can be defined as one or many. With a link type of cardinality one, only one link of this type can exist at a given time from a given object. With a link type of cardinality many, several links of this type can exist from the same object. In this case, each link must be designated by a sequence of unique key attribute values.

The category of a link type can be defined as existence, composition, reference, designation, or implicit. It defines integrity constraints as explained in the sequel.

In the OMS database, an object can only be created as destination of a link whose type is of category existence or composition. Links whose type is of one of these categories can also be created leading to existing objects. Conversely, an object is implicitly deleted when its last incoming link of category existence or composition is deleted.

A link whose type is of category reference does not allow the creation of an object (as its destination) but prevents the deletion of its destination object.

A link whose type is of category designation does not prevent the deletion of its destination object.

A link whose type is of category implicit is always implicitly created or deleted (as a side-effect of the creation or deletion of its reverse link in a relationship). The main role of these links is therefore to reverse links whose type is of category existence, composition, or reference in relationships. In reality, links whose type is of any category but designation are always in relationships. Only links whose type is of category designation are not implicitly reversed.

When a link is created, a certain degree of stability may be required on its destination object. This stability property can only be specified for links whose type is of category existence, composition, or reference. A link whose type has the stability property set to true stabilizes its destination object, i.e. prevents any modification of its contents, its attributes, and its outgoing links whose type is of any category but implicit. Hence, only links whose type is of category implicit can still be created or deleted from a stable object.

When a link type of category composition is defined, the exclusiveness (or non-sharability) of the destination objects...
of its instances may be required. This is done by setting the exclusiveness property to true.

The duplication property of a link type determines whether or not the links of this type are to be copied when their origin objects are copied. This property can be specified for links whose type is of category existence, composition, reference, or designation.

Two link types can be bound together in order to form a relationship type. According to the cardinalities of the link types, one to one, one to many, and many to many binary relationship types can be defined.

4.2 Concepts of Composite Entities and Object's Contents

The second requirement is twofold: to provide mechanisms for defining and managing composite entities (being able to define and manage its internal structure down to the level of fine grains), and in addition to this, to support uninterpreted data (i.e. data merely being held by the database system without any interpretation of its value or internal structure). This requirement concerns the issue of granularity support. Ideally, a software engineering database system should support very huge grains as for example a whole software system describing and managing its internal structure down to the level of nodes of the abstract syntax tree of its source programs. For reasons of space and performance, the level of granularity suggested for the PCTE+'s OMS is for instance paragraphs in a document. Simultaneously, it is necessary to be able to store uninterpreted data as for example object code of programs or whole documents. The PCTE+'s OMS satisfies this requirement by means of two concepts: composite entities and object's contents.

In the OMS data model, a composite entity is defined as the collection of objects and links constituted by:

- a given object (used to designate the whole composite entity),
- the set of objects which are in the transitive closure of links starting from the given object and whose types are of category composition,
- the links which are starting from the given object and from the objects of the set defined above.

The objects of a composite entity are called its components. The given object itself is called the root component of the composite entity. It is used to designate the composite entity. OMS operations that affect as a whole the objects and links which constitute a composite entity are provided. These operations are also applicable to the limit case of a composite entity made up of only one object (i.e. a single entity).

As can be seen, the concept of composite entity is mainly based on the notion of links of category composition. Hence, if an object is created as destination of a link whose type is of category composition, its destination object is considered to be a component of its origin object (which designates a composite entity). If a link whose type is of category composition is created leading to an existing object, its destination object is adopted as a new component of its origin object. In this way, components can be shared between different composite entities. A mechanism is also provided for enforcing exclusive (non-sharable) components. It is the exclusiveness property. If a link is an instance of a link type of category composition having the exclusiveness property set to true, its destination object is an exclusive (or non-sharable) component of its origin object. In this way strictly hierarchical structures can be created and enforced within composite entities.

As seen in the preceding section, when a link is created, a certain degree of stability may be required on its destination object. In order to stabilize a composite entity, a link whose type has the transitive stability property set to true must be created leading to its root component. By means of its transitive effect, all of its components are stabilized.

The OMS data model supports uninterpreted data by means of the concept of object's contents. This is similar to the concept of file in operating system.

The kind of contents of an object depends of its type. The predefined object types having a specific contents are: file, pipe, message queue, and device. All object types which are descendents of one of these predefined types have the same kind of contents. The other object types define objects without contents. Only one kind of contents can be defined for an object type.

4.3 Concept of Versions of Composite Entities

The third requirement is to provide mechanisms for version management. The PCTE+'s OMS satisfies this requirement by means of three concepts: composite entities, graphs of versions of composite entities, and operations for managing versions of composite entities. Versions of single objects are treated as a special case, that is as a composite entity with only one component, the object.

A composite entity defines a collection of inter-related objects. Versions of composite entities (and as a special case single objects) can be created, manipulated, and accessed by means of OMS operations for managing versions. The history of how the successive versions of a composite entity have been created is kept by means of a version graph with an explicit representation of the history information.

When a new composite entity is created as a new version of a given composite entity, a "predecessor-successor" relationship is created between the corresponding components of the two composite entities in order to maintain the history of these versions.

In order to maintain the consistency of the history of versions, the predecessor link type is characterized by the transitive stability property in such a way that a version cannot be changed while a successor exists.

A new version is created from an existing version, thus leading to a tree of versions. It is also possible to add new predecessors to an existing version in order to represent the effects of a merge tool. The general form of the version graph is therefore a directed acyclic graph.
The graph of versions does not impose a naming policy for accessing any version though, of course, any version can be accessed by navigation along the "predecessor-successor" relationships. Thus, the policy defining how successive versions can be named is not predefined by the OMS. A software engineering environment builder can decide his or her own policy which may be supported by tools. Naming policies can be implemented by means of tool-defined structures of link types. The preferred link name mechanisms can be used in order to define the usual notion of preferred version. The preferred link names are described by a pattern. For instance, the \( ^{-}.+ \) pattern specifies a link key name with two attributes: the symbol \( ^{-} \) matches any string attribute and the symbol \( ^{+} \) matches the maximum value among the values of integer attributes of links.

4.4 Concept of Type Specialization

The fourth requirement suggests the need of inheritance mechanisms for defining new types from existing ones while maintaining the compatibility with them. The PCTE+'s OMS satisfies this requirement by defining new types as specializations of existing types. Additionally to its specific characteristics, the new defined type inherits all of the characteristics of its supertypes.

Thereby, in the OMS, a new object type is always defined as a subtype of (i.e. a specialization of) one or more other types. In addition to its applied link types and attribute types, the new defined type inherits all of the applied link types, attribute types, and the kind of contents of its supertypes. A predefined object type, called "object", is the common ancestor type of all object types. Other predefined object types (which in turn are subtypes of "object") are: file, pipe, message queue, volume, device, process, and activity. A new defined type can be subtype of only one of these predefined types (which defines the kind of contents).

The structure formed by these "subtype" relationships between types constitute a connected direct acyclic graph having only one root, the type "object". Multiple inheritance is therefore provided.

4.5 Concepts of Schema Definition Sets and Working Schemas

The fifth requirement is to support the definition of multiple views of the common database by different tools and users. The PCTE+'s OMS satisfies this requirement by means of two concepts: Schema Definition Sets (SDS) and Working Schemas.

The OMS data model defines the organization of type definitions (either object, link, relationship, or attribute types) into Schema Definition Sets (SDS). Each SDS gathers a set of related type definitions which constitute a subset of the database schema. Examples of SDSs are the set of definitions that are necessary for all users working on a given project, the set of definitions that are specific to a given user, and the set of definitions that are specific to a given tool. Each SDS is a complete, consistent, and self-contained subset of the database schema. Thereby, the overall OMS database schema is defined by the union of all defined SDSs. Hence, unlike conventional database systems, the OMS database schema can easily change in time by the addition of new SDSs or the modification of existing ones. In this manner, new tools can be easily incorporated in software engineering environments built on PCTE+ as the specific set of type definitions that they need can be added in the form of new SDSs or modification of existing ones.

An important feature of the SDSs is the sharing of type definitions among SDSs. This is provided by the "importation facilities". The importation of a type definition from an SDS into another makes this definition visible in both SDSs. In this way, type definitions can be shared among several SDSs.

SDSs and types are represented as objects in the OMS database. The set of objects representing types is called the metabase. The types of these objects are themselves described in an SDS called "metasds", thus providing a self-description schema.

With each tool (and each user via a tool) is associated a view of the OMS database which is called its working schema. A working schema is a logical view of the OMS database, defined in such a way that the database can be shared among different tools, each tool accessing the common database through the working schema it is associated with. The common database then becomes the synthesis of all of the working schemas associated with the tools (and users via tools). Furthermore, a tool can change its working schema dynamically.

A working schema is defined by a list of SDSs. The view of the database defined in a working schema is given by the set of type definitions which is the union of all of the type definitions in the SDSs of the list. For instance, objects of a given type that is in one or more SDSs of the list are seen having as attributes and possible links the ones whose types are applied to the object type in the SDSs of the list.

Working schemas allow to see from different points of view the common OMS database. Therefore, it provides the basic mechanism that allows integration of different tools that come from different sources. These tools can be integrated by sharing and exchanging data via the common database which is viewed in each tool's individual manner.

4.6 Mechanisms for Concurrency and Integrity Control

The sixth requirement is to provide mechanisms for concurrency control. The PCTE+'s OMS satisfies this requirement by providing lock mechanisms for ensuring the consistency of OMS data access operations by controlling the synchronization of concurrent operations on the same data.

The seventh requirement is to provide integrity control by transaction mechanisms. The eighth requirement asks in addition to support nested transactions. The PCTE+'s OMS satisfies these requirements by means of the concept of activity which includes the one of transaction.

An activity is the framework in which a set of related OMS
operations takes place. An activity, by its class, allows to define the required degree of protection (and so integrity) of the data accesses.

There are three classes of activities:

- unprotected activities: they do not require their data accesses to be protected from other concurrent activities;
- protected activities: they require their data accesses to be protected from other concurrent activities for the stability of the input data and to preserve output data from untimely overwriting, however the effect of the activity as a whole is not required to be atomic;
- transaction activities: they are protected and atomic activities.

Activities may be nested. Nesting is achieved by starting an activity within an existing one. It is worth noting that in the case of nested transaction activities, a nested transaction activity may commit or abort without implying the ending of its enclosing transaction activity.

An activity always has a set of acquired resources and associated locks which are held on these resources.

A resource is either an object with its contents and its attributes, or a link with its attributes.

Lock mechanisms ensure the consistency of OMS data access operations by controlling the synchronization of concurrent operations on the same resources.

Locks are characterized by lock modes. They are formed by meaningful combinations of the kind of the required protection (unprotected, protected, and transaction) and the kind of the required data access ("read", "write" (including "read"), and "delete" (including "write").

Activities and locks are represented in the OMS database. The former is represented as objects and the latter as links.

4.7 Mechanisms for Discretionary Access Control

The ninth requirement is to provide mechanisms for access control. The PCTE+’s OMS satisfies this requirement by means of the concept of security. It is defined as the prevention of unauthorized disclosure, amendment or deletion of information held in the OMS database. Both mandatory and discretionary security controls as defined in [TCSEC 83] are provided.

Discretionary security control is the means of restricting access to objects in the OMS database based on the identification of users and/or groups to which they belong. Mandatory security control is the means of restricting access to objects in the OMS database based on the sensitivity of the information contained in the objects and the formal authorization of users to access the information.

The conventional notion of access control in database and operating systems corresponds to the one of discretionary security control. In PCTE+ this kind of access control is enforced as follows. Each OMS object is associated with an access control list which defines which accesses are permitted to the object for designated users, groups, or tools. Access control lists are expressed in terms of elementary access rights (read, write, append, navigate, execute, control, etc.) which are explicitly granted or denied to designated individual users, user groups, or tool groups.

4.8 Support for Multiple Programming Languages

The tenth requirement is to provide support for tools written in different programming languages. The PCTE+’s OMS satisfies this requirement by defining the database concepts in a language-independent way and specifying the interface of operations in different programming languages.

Up to now two interface specifications were defined: one in the C programming language and the other in Ada’d [PCTE+ Ada & C 88]. The C language specification of the interface is defined as a set of functions that may be called by tools. The Ada language specification is defined as a set of packages providing procedures and functions that in the same way may be called by tools.

4.9 Support for Distributed Architecture and Database

The eleventh requirement is to support a hardware architecture based on a network of (possibly heterogeneous) workstations and mainframes. The PCTE+’s OMS satisfies this requirement by providing a data model whose design took into account the constraints of a distributed hardware architecture. In fact, the OMS data model was designed to be implemented in local area network of possibly heterogeneous workstations and mainframes.

The twelfth requirement is to provide transparent distribution of the database over the network of workstations and mainframes. The PCTE+’s OMS satisfies this requirement by providing a transparently distributed OMS database. The OMS database is distributed among workstations on a network in such a way that users can access the objects, links, and attributes held in the database without having to be aware of their location.

The distributed nature of the OMS database can contribute to increase availability of information by allowing any workstation to support a set of objects that are local to that workstation. Furthermore, mechanisms are provided for managing replicated objects (i.e. several copies of the attributes, contents (if any), and links of such objects existing throughout the network of workstations).

5. Conclusions and Future Work

In summary, in this paper we primarily presented the software engineering database system that is at the heart of the PCTE+ tool support interface: the PCTE+’s OMS.

Emphasis was placed on the concepts and mechanisms pro-
vided by the PCTE+'s OMS. They were designed taking into account the database requirements issued from past experiences in building tool support interfaces and software engineering environments, in particular the PACT environment on PCTE. Therefore, we believe that these concepts and mechanisms are well-suited for describing and managing the information produced and manipulated by software engineering environments. Furthermore, the distributed nature of the PCTE+ OMS is an essential feature for supporting large-scale software engineering environments.

The PCTE+ OMS extends the PCTE's OMS [Gallo et al. 86, [PCTE Ada & C 88] with several new concepts and features. They include: duplication, exclusiveness, and transitive stability properties, the designation category, enumeration and real value types for attributes, composite entities, version mechanisms, multiple rather than simple type inheritance, nested transactions within processes (i.e. program execution), lock modes with "delete" data access, a new model for discretionary and mandatory access control, a self-description schema, and a self-referential model where activities and processes are represented as objects.

The PCTE+ tool support interface, including the PCTE+'s OMS, was specified during the definition phase of the PCTE+ project. Specifications were produced in the form of Ada and C bindings. Two prototypes of PCTE+ are planned as part of the assessment phase of this project, on distributed Unix and distributed VMS hosts respectively. The one on Unix is expected to be ready by the end of 1990. Both will follow the so-called "black-box" approach where the PCTE+ interfaces will be implemented in terms of libraries on top of the operating systems. The assessment phase will also include the integration of a wide range of tools, as well as an assessment of the suitability of PCTE+ as a tool support interface for the development of highly critical, software intensive systems. More details on ongoing projects of the PCTE initiative, including the PCTE+ project, can be found in [Campbell 89].

Future work is primarily concerned with the support of software processes. In the ALF project, work is being done to enhance the next generation of PCTE+ with features suitable for supporting software process description and enactment [Griffiths et al. 89]. Issues of triggering are also being addressed [Oquendo 89] to enhance the next generation of PCTE+ and further to make the PCTE+'s OMS active.

The European Computer Manufacturers Association (ECMA) set up a technical committee (TC33) with the objective to decide on an ECMA PCTE. PCTE+ was taken as the start point for ECMA standardization, to be followed in 1991 by preparations for ISO standardization.

Acknowledgements

We should like to thank Ian Campbell (the PCTE+ project manager) for his continual help and advice, and Ken Hayter

References


