Integrating Software Tool Communication Within An Environment

Peraphon Sophatsathit and Joseph Urban

Arizona State University

Abstract

The proliferation of software tools has simplified the tasks of software development. As demands and costs for improving software quality escalate, more powerful development support environments are required to carry out the tasks. However, some obstacles have precluded software developers from attaining those demands and objectives, such as tool use, training, productivity metrics and measurement, and maintenance problems. Various CASE tools are employed to aid in the development process. In many cases, these tools are developed by independent vendors to run on different platforms. The results of these tool applications are therefore incompatible.

To ease the burden of tool discrepancies and incompatibility, standardized public tool interfaces were proposed to permit tools developed by different vendors to coexist, interoperate and be able to transport across computer systems. This paper surveys some of the predominant Integrated Project Support Environments (IPSE) to establish a viable architectural framework and requirements for a Virtual Software Communication System (VSTC) prototype.

Index Terms: SEE, CAIS, PCTE, VSTC, rapid prototyping, tool-to-tool communication.

Introduction

Software tool communication is a means for different tools to share, exchange, and manipulate information with minimal conversion. The objective of tool communication is to provide a common interface for various software tools created by different vendors that run on different platforms to interoperate with one another smoothly and effectively. Sullivan [19] defines an integrated environment as:

> "At the requirements level, an integrated environment is a collection of user-level software tools and a collection of automatically maintained relationships among these tools. Relationships tie tools into cohesive environments, freeing users from having to manually integrate the tools."

Charette [7] points out problems with software development:
- tools and methodologies that work well for one project may not be applicable to others,
- too much time and effort spent on documentation, and
- lack of understanding on management part.

He then defines the definition of a software engineering environment (SEE) as follows:

> "The process, methods, and automation required to produce a software system."

This definition encompasses the following aspects:
- the establishment and use of sound engineering principles in order to economically obtain software that is high quality, reliable, and works efficiently on real machines;
- a systematic approach to the development, operation, and maintenance to accomplish high quality software;
- a set of support facilities that aid in software development in a productive and cooperative manner. A special automated tool set, known as Computer-Assisted Software Engineering (CASE), serves in an important role in many software development projects; and
- a standardized methodology and consistent transformation of software being developed throughout its life cycle.

In order to understand the notion of what constitutes tool communication, a couple of software process fundamentals are described to establish the requirements and relationships of these processes to SEEs and the tools that run under SEEs, namely, *Integrated Project Support Environment* and *Development Support*.

Integrated Project Support Environment (IPSE)

Integrated Project Support Environment (IPSE) was brought about to help solve software development problems in recognition of the software crises. However, as mentioned earlier, the growth, and proliferation of tool development and usage created wide arrays of project incompatibilities, data sharing, and tool communication. Charette [7] points out a number of shortcomings of current software development practices:
- high cost of software,
- variation in practice, and
- need for increased productivity.

Another important software crisis that must also be taken into account is:
- reliability and quality of the software product and environment.

Efforts to provide some cooperation among different SEEs were initiated, in particular, the Common APSE Interface Set (CAIS) and the Portable Common Tool Environment (PCTE) which represent two most prominent systems. The contribution brought about by these systems is extensive, such as:
- *Openness* that offers the ability to incorporate a variety of new methods, tools, and processes,
- *Scope of integration* that defines the extent to which the

components are integrated to provide a homogeneous environment,
- *Scope of support* that specifies the extent of activities supported by the environment from requirements to maintenance, and
- *Guidelines for tool interface and communication standards* that provide a framework for tool portability and integration.

Oddy [15] describes four classes of SEE models based on the orientation and flow of information, namely,
- *Language-centered environments* which support a particular language,
- *Structure-oriented environments* which support language-independent structural manipulation,
- *Toolkit environments* which provide language-independent tool support for various development activities, and
- *Method-based environments* which provide tools that support specification and design methods.

In essence, he categorizes the first two classes as PSE, and the latter two IPSE, to distinguish the two environments. Efforts have been made to improve existing IPSEs, ranging from single user, tightly integrated environment settings, such as Borland's Turbo family, to multi-user, loosely integrated environment settings, such as Software Techniques of Engineering Project (STEP), Precise Interface Control (PIC), and Microelectronics and Computer Technology Corporation (MCC). The Development support section describes some major components and characteristics of the related work, namely, the CAIS, the PCTE, and the Unix® operating system. The VSTC section lists a set of requirements for tool-to-tool communication, as well as some development details.

Development support

Ideally, common tool communication support should encompass the following:
- multiple user roles;
- multiple environments;
- multiple languages;
- common and consistent interface operations;
- knowledge-based methodology support (rules describing the methods / procedures to be applied to software development);
- extensibility, which implies that
  - customization of user interface,
  - customization of database, and
  - compatibility with existing and new software tools;
- tool integration that offers
  - consistent user interface (operational and visual consistency), and
  - tailorable capability which leads to conformant user interface of foreign tools;
- database
  - transaction processing,
  - Database Management Systems (DBMS),
  - query language capabilities, and
  - database access utilities for tool integration; and
- software reuse.

Other issues such as user-friendliness, security and integrity, management support, methodology, language support, and documentation, are also essential to usability and overall performance of the common tool interface system.

Related work

Numerous efforts have been made in the areas of SEEs, IPSEs, and the Software Factory Environment for the engineering process [11] which includes methods, development rules, standards, and control of data management, engineering technology, as well as product assurance, to integrate and standardize software tools, and their corresponding interfaces. Some of these efforts are CAIS-A, PCTE+, the EAST Eureka Project which is based on the PCTE [1], ECLIPSE [3], Open Software Foundation (OSF), and Ferranti Computer System Ltd. The following sections discuss two large-scale IPSEs and an early software development environment, namely, CAIS / PCTE and Unix, respectively.

The Common APSE Interface Set (CAIS)

The inception of the Ada effort, initiated by the U.S. Department of Defense in 1975, has brought about numerous initiatives in the area of IPSE and SEE. In 1980, a document produced by Buxton [5], known as the Stoneman report, established a framework for a common programming language environment. Two procurements were started, namely, the Ada Language System (ALS) by the Army and the Ada Integrated Environment (AIE) by the Air Force. Unfortunately, the divergent approaches at the ALS Kernel Ada Programming Support Environment (KAPSE) and the AIE KAPSE interfaces created significant discrepancies in the portability and interoperability objectives. This resulted in a team formed by a tri-services agreement which defined more specific requirements for an Ada Programming Support Environment (APSE) and its associated layered architecture. This team was known as KAPSE Interface Team (KIT). Added later was the KAPSE Interface Team from Industry and Academia (KITIA). The KIT / KITIA, under the Ada Joint Program Office (AJPO), developed the CAIS as the basis for integrated APSEs to address the issues of tool portability and project databases between different host computers.

The primary objective of the CAIS was to establish a set of standards which support Ada software transportability and sharing of tools across different APSEs. A common set of interfaces was therefore necessary to facilitate machines and operating systems independence, and to accomplish tool interoperability and transportability by means of inter-tool data standards, such as MIL-STD-1838A [6], which was one of the most widely referenced documents during the early development of the CAIS.

Some basic features of the CAIS are:
- *Object Management Facilities* which are based on the Entity-Relationship-Attribute (E-R-A) model that includes create, delete, copy of objects, relationships between objects, and their attributes;
- *Typing of Objects* which makes the structure of database objects visible to all tools;
- *Flexible View Mechanism* which allows different views of the same object by different tools to enhance data sharing;
- *Transaction Facility* which supports atomic execution by different tools;
- *Data Monitoring and Triggering Facilities* which monitors and automatically triggers the execution of a process;
- *Process Management Facilities* which offers synchronous or asynchronous process invocation and control;
- *Flexible IO Mechanism* which allows unified file IO and Inter-Process Communication operations; and
- *Distributed Implementation Support* which facilitates programmatic control of various object identifications and locations in heterogeneous computer networks.

These features can be summarized into the following categories:
- Node Model,
- Processes,
- Input and Output,
- Transactions, and
- Distribution.

Details of CAIS requirements and implementation can be found in [6, 14].

The Portable Common Tool Environment (PCTE)

Lyons [13] describes three factors associated with the subject of public tool interface (PTI):
1) The use of SEEs and their associated tools and methods,
2) Standardization of ALS, and
3) The impact of the Stoneman Requirements for the APSE's document.

These factors led to the development of early PTIs, such as the Ada Language System (ALS), Ada Integrated Environment (AIE) in the U.S., Portable Ada Programming System (PAPS), Minimal Chill / Ada Programming Support Environment (MCHAPSE) in Europe. The start of the ESPRIT programme of the Commission of the European Communities (CEC), led by six European computer manufacturers in 1983, initiated a project entitled "A Basis for a Portable Common Tool Environment." Subsequent evolution of the PCTE to various projects such as PCTE+ and Pact environment are described in [4, 21].

The PCTE is an infrastructure through which different tools can communicate. The PCTE is aimed at providing a PTI environment which aids in the development, integration, and execution of software tools. The objectives of the PCTE are:
- *Generality* which facilitates a wide range of applications and development methods;
- *Flexibility* which provides simple, yet powerful, services to suit user needs;
- *Homogeneity* which preserves logical consistency between tools and data, as well as uniform system and user interaction;
- *Portability* which maintains system specific independence; and
- *Compatibility* which supports existing architectures and standards.

The basic mechanisms of the PCTE to carry out the above objectives are:
- Object Management System,
- Processes or Execution Mechanisms,
- Concurrency Mechanisms,
- Communication Mechanisms, and
- Distribution Mechanisms.

These are elucidated in Lyons [13].

Unix

According to Ritchie and Thompson [18], Unix is a general-purpose, multiuser, interactive operating system that offers:
- A mountable hierarchical file system,
- Compatible file, device, and inter-process input / output constructs,
- Asynchronous process creation,
- User-level command language interface,
- System utilities, and
- High degree of portability.

The infrastructure of Unix is built on the *kernel* and the command language interpreter or *shell*. Tools are independently developed, installed, invoked and interact with one another through built-in shell functions, such as pipe, redirection, filter, or through kernel mechanisms, namely, inter-process communication, semaphore, and message-passing. The Unix environment consists primarily of:
- shell,
- Configuration Management utilities, e.g., make, Source Code Control System, and Revision Control System,
- editors,
- compilers,
- libraries,
- debuggers, and
- utilities.

Flexibility and portability are the strengths of the Unix environment in spite of the implementation variations on different platforms, such as System V, Berkeley Software Distribution, Santa Cruz Operation Unix, Advanced Interactive Executive, Ultrix, and Xenix. Applications written in one platform can be transported to others with minimal changes. In addition, many recent efforts have been carried out in an attempt to standardize and to make the Unix environment more user-friendly and integrated, for instance, Portable Operating System Interface for Computer Environments (POSIX), and International Standard Organization (ISO). Moreover, the introduction of Window Systems, in particular, X Windows™, OPEN LOOK™, and Microsoft Windows™, brought about new paradigms and meanings of PSE / SDE to the Unix environment. The Application Program Interface, and Graphical User Interface, are a few predominant examples.

Despite the abundant support facilities and recent development efforts, the Unix environment is not ideal for a SEE in that some features, such as fork, exec, and signals, may be hard to implement on other environments, making software tools difficult to transport to other platforms. One of the main limitations that restricts common interface support to attain a tool communication capability is the file system. The Unix file system is too primitive to serve as the central information repository that a DBMS can accomplish. This simplicity makes it difficult to establish an environment to handle the evolving requirements of IPSE, object-oriented design, and high-level abstraction paradigms, such as generalization and specialization, abstract data type, and inheritance. Nevertheless, Unix environment is still regarded as the first-generation IPSE [9].

Virtual Software Tool Communication (VSTC)

Virtual Software Tool Communication (VSTC) is proposed to support the aforementioned notion of tool-to-tool interoperability. The most prevalent conventional tool-to-tool communication approach is by means of conversion programs written specifically for the host and target platforms. No provision has yet been made to cope with transportability of foreign software without the use of special conversion programs. VSTC, on the other hand, employs a portable OMS concept as its kernel and an interface subsystem tailored to accommodate system-specific environments. This process is shown in Figure 1, which illustrates the underlying architecture of the VSTC reference model.

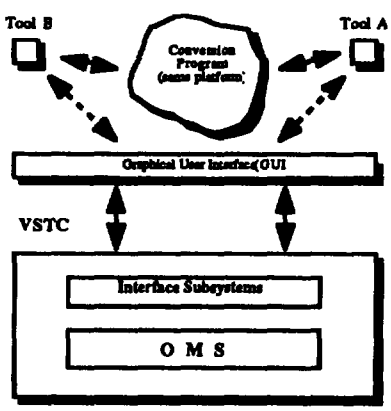

Figure 1. Architecture of the Reference Model

The next three sections describe the characteristics of the proposed VSTC system, the development of the reference

architecture and its corresponding requirements specification, and the implementation of an ad hoc model with some criteria for the evaluation of the VSTC system, respectively. The benefits and future research prospects of the VSTC are summarized in the last section.

Characteristics

To establish a framework for an efficient tool interoperating interface, the VSTC focuses on the following aspects of common software tool communication:

1) A heterogeneous database which provides centralized control of operational information during software development; and
2) An integrated support environment which is an abstract environment that provides a standardized interface and a set of common services to support software tool cooperation.

The *heterogeneous database* provides centralized control and support facilities for operational information during software development and maintenance. System specifications, design documents, source code, test data, maintenance plan, documentation, quality assurance plan, as well as viewing mechanisms, can be transported and interoperated on different data objects through a standardized interface. Thus, this heterogeneous database offers:

- A mechanism for representing data objects which is analogous to the Object Management System (OMS) of the PCTE, and the Entity Management System (EMS) of the CAIS;
- A common object library system based upon the notion of object-oriented design and object management system [2];
- Information repository and sharing compatibilities between different applications, tools, methodologies, and projects, such as source code, metrics, test cases, and object typing.

The *integrated support environment* establishes an abstract environment to provide a set of predefined objects and operators. These objects, in turn, allow users to create user-defined composite objects to suit their needs. Moreover, the environment should provide limited facilities that enable users to access and interact with the information base. These facilities are:

- An interface management system that facilitates coverage of the development process to ensure smooth transition from one tool to another without undue interference or overlapping roles. The fundamental concept is based on programs (modules) cooperating through calls and data sharing, what functions each module provides, and the interconnection description of the resource flow between modules in the system;
- Tool cooperation which is accomplished by means of a set of common services that allow tools to communicate in the abstract environment. Such services include applications of user-defined typing definitions, creation of new types and establishment of definitions of non-local tools.

Development

The system is defined to deal with specific software tool environments as a prototype to demonstrate the theoretical notion of tool communication. The system is defined to provide a number of essential components similar to the aforementioned large-scale systems, namely, the CAIS and the PCTE. Moreover, a number of recent research efforts in the area of object-oriented databases are incorporated into the system to support the construction and operation of the OMS.

The following sections describe an architecture of a Reference Model and a set of basic functional requirements based on the European Computer Manufacturer Association (ECMA) Reference Model [10].

The reference architecture

Figure 1 illustrates the reference architecture of the proposed VSTC system. This model is based on the following assumptions:

- The VSTC is a system independent software system to run on standard Unix tools. Thus, the VSTC may not run on any subsets or variations of Unix operating system variants that are not fully compatible with the standard Unix operating system;
- Interface mechanisms for inter-module connection and control can only be accomplished through static type checking [17]. There is no provision for run-time support to minimize system overhead and improve performance efficiency;
- Primitive objects are predefined and cannot be altered;
- Data Dictionary format is predefined and of fixed size for optimal efficiency and execution speed;
- The working schema only permits one-to-one and one-to-many relations. Many-to-many relations are not supported;
- Nested queries are not supported by the system, only simple queries are to be implemented; and
- Views are to be restricted to the object level for efficiency reasons. Package viewing will not be supported.

System requirements specification

The VSTC shall provide support for the following requirements:

- Relationships between interface and subsystems, such as level of formalism, connection and control, and communication interface;
- OMS, including object definition and typing, composite object, and inheritance [16];
- OMS Services, such as data access and retrieval from the object repository, granularity of change, and library primitives;
- Data Query and Manipulation Common Service (DQMCS) [12, 20], such as Data Dictionary and DDL, working schema, query processing, and view mechanisms;
- Packaging mechanisms and package reuse;
- Tool-data import and export capabilities; and
- Graphical User Interface.

A VSTC OMS model

In order to test the validity of the above requirements and to undertake some major component risks into account at the earliest possible stage, an *evolutionary rapid prototype* for the OMS was devised as an experimental system such that it would:

- demonstrate critical functions of the OMS after a minimal amount of effort,
- serve as a means to provide an overview, but not necessary represent a real VSTC system,
- be flexible to build, modify, enhance or discard if need be,
- support the verification of the stated requirements.

Rapid prototyping

The process of prototype development for the OMS was broken down into four stages represented by four different diagrams, namely, *the context diagram, the essential functions diagram, the E-R diagram,* and *the control-flow graph.*

The *context diagram* contains the processes under consideration, along with major input and output transformation processes. The *essential function diagram* encompasses major functions performed by the OMS. The *E-R diagram* denotes the structural relationships between various objects manipulated and stored by the OMS. The

control-flow graph depicts a preliminary OMS prototype module calling sequence.

Criteria

To determine the applicability of the prototype as a means to validate the requirements, an evaluation method was required to measure the prototype created earlier. Connell and Shafer [8] outline a set of criteria to assess a project plan for rapid prototyping, namely, *approach justification, goals, scope of effort, development tools, user responsibilities, deliverables, and preliminary schedule.* Partial criteria were adopted to assist the prototype evaluation process, i.e., approach justification, goals, and scope of effort.

• *Approach justification* There are a number of advantages to this rapid prototyping method. First, it offers a quick overview of the overall requirements of the OMS from different perspectives, namely, from the OMS context standpoint, from the essential functions standpoint, from the control flow standpoint, and from the entity (data) relationship standpoint. Second, essential requirements can be analyzed and validated incrementally which, in turn, lead to the establishment of functional specifications of the OMS. Third, it provides a concise and easy to modify high-level conceptual model that will eventually evolve into a working reference model. Finally, details in object attributes, functionalities of object library, packaging constructs and techniques can be avoided at such an early stage to concentrate on more important issues at hand. That is to say, fine tuning of various design approaches, performance criteria, and implementation issues can be postponed until the appropriate stage.

• *Goals* As mentioned earlier, the objectives of the development of the prototype were to validate the requirements of the OMS, and to arrive at the functional specifications for all major components of the OMS,. Moreover, component sharing and reuse were identified early from the outline of the prototype. Such identification enables a derivation of an empirical system configuration, which subsequently will lead to an efficient system design and implementation.

• *Scope of effort* Some anticipated methodologies, tools, DBMS schemas, functional considerations, system component interactions were identified during the analysis of the prototype. Decisions on issues such as component boundary, inclusion of other related components or activities, as well as component constraints were visible and partially established at the outset. Such early unravelling of constraints and limitations lent themselves to subsequent derivation of system specifications.

Table 1 summarizes the resulting evaluation of the requirement prototype using the first three criteria suggested by Connell and Shafer [8].

Summary and Future Research

The research was an attempt to establish a small-scale, demonstrable working prototype which can be used as a laboratory research vehicle for a development environment. The concepts are based on the use of a common repository to facilitate sharing, exchanging, and transferring of data by standard and non-standard tools that access and manipulate these data. The goal is to provide a base system which will enable users to import and export software tools across computer systems with minimal efforts and reconfiguration. The system will enable the software developer to exploit the environment support across a wide array of computer systems based on a standardized interface and conforming requirements.

The final demonstration feasibility in the form of a prototype will be delivered as a software engineering laboratory support system. The prototype is intended to be a virtual layer which will not involve as closely with the operating systems activities as the CAIS and the PCTE. The paradigms employed in the proposed system, namely, the abstract environment, the interface mechanism, and the object-oriented design can also be extended to other areas of integrated support and development environments or as a model for future research. For instance, configuration management support, transaction management, and Interface Command Language Interpreter.

OMS req / criteria	Approach Justifica.	Goals	Scope of Effort
Data access / retrieval from repository	obtain a macroscopic view of the OMS func. on data obj	methods by which object info. access and ret are carried out	access control through message manager
Granularity of change	E-R relationship mappings	rules to stipulate structure and content modification	accomplish through inheritance mechanism
Library Primitives	methods of composite objects, and package formation	organize and manipulate primitive objects	only accessed, manipulated via obj mgr; packages, user-defined object types are formed based on obj relationship with primitives
DD and DDL	quick overview of the roles of DD and DDL	identify info. to be stored by DD for OMS services	interact only with query and object routines through DD front-end
Working Schema	analyze schema manipulation techniques	schema and relationships construction and modification	directly managed by schema mgr isolate from other DQMCS functions
Query	overview of query processing path	OMS query processing method	consult DD for all data objects needed
View	a mechanism to study how to specify multiple interfaces for obj.	info. hiding and access control, while supporting schema browsing	restricted levels of display and access of obj granularity by predetermined working schemas

Table 1. Prototype Evaluation

References

[1] K. H. Bennett, *Software Engineering Environments: Research and Practice*. Chichester, New York: Halsted Press, 1989.

[2] G. S. Blair, J. Malik, J.R. Nicol, and J. Walpole, "A synthesis of object-oriented and functional ideas in the design of a distributed software engineering environment," *Software Engineering Journal*. May 1990, vol. 5, no. 3, pp. 193-204.

[3] F. Bott, *ECLIPSE-An Integrated Project Support Environment*. London, United Kingdom: Peter Peregrinus Ltd, 1989.

[4] G. Boudier, F. Gallo, R. Minot, and I. Thomas, "An Overview of PCTE and PCTE+," *Proceedings of the ACM SIGSOFT / SIGPLAN—Software Engineering Symposium on Practical Software Development Environments*. November 1988, vol. 13, no. 5, pp. 248-257.

[5] J. N. Buxton, *STONEMAN: Requirements for Ada Programming Support Environments*. The United States Department of Defense, Washington, February 1980.

[6] "Introduction to CAIS," *Common Ada Programming Support Environment (APSE) Interface Set*. MIL-STD-1838A, 30 September 1989.

[7] R. N. Charette, *Software Engineering Environments—Concepts and Technology*. New York, New York: Intertext Publications, Inc. (McGraw-Hill Book Company), 1986.

[8] J. L. Connell and L. Shafer, *Structured Rapid Prototyping—An Evolutionary Approach to Software Development*. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1989.

[9] P. W. Dell, "Early experience with an IPSE," *Software Engineering Journal*. November 1986, vol. 1, no. 6, pp. 259-264.

[10] ECMA Technical Committee (TC33)—ECMA TR / 55, *A Reference Model for Frameworks of Computer-Assisted Software Engineering Environments*. European Computer Manufacturers Association (ECMA), 114 Rue du Rhone - CH - 1204 Geneva (Switzerland), December 1990.

[11] M. W. Evans, *The Software Factory—A Fourth Generation SEE*. New York: John Wiley & Sons, 1989.

[12] B. Hailpern and H. Ossher, "Extending Objects to Support Multiple Interfaces and Access Control," *IEEE Transaction on Software Engineering*. November 1990, vol. 16, no. 11, pp. 1247-1257.

[13] T. G. L. Lyons, "The public tool interface in software engineering environments," *Software Engineering Journal*. November 1986, vol. 1, no. 6, pp. 254-258.

[14] R. Munck, P. Oberndorf, E. Ploedereder, and R. Thall, "An Overview of DOD-STD-1838A (proposed), The Common APSE Interface Set, Revision A," *Proceedings of the ACM SIGSOFT / SIGPLAN—Software Engineering Symposium on Practical Software Development Environments*. November 1988, vol. 13, no. 5, pp. 235-247.

[15] G. C. Oddy, "Systems and Software Environment or Factory?," *Software Engineering Environments: Research and Practice*, K. H. Bennett (Editor). Chichester, New York: Halsted Press, 1989, Chapter 3, pp. 29-44.

[16] M. H. Penedo, E. Ploedereder, and I. Thomas, "Object Management Issues for Software Engineering Environments—Workshop Report," *Proceedings of the ACM SIGSOFT / SIGPLAN—Software Engineering Symposium on Practical Software Development Environments*. November 1988, vol. 13, no. 5, pp. 226-234.

[17] R. Prieto-Diaz and J. M. Neighbors, *Module Interconnection Languages: A Survey*. Department of Information and Computer Science, University of California Irvine, Irvine, CA 92717. August 1982.

[18] D. M. Ritchie and K. Thompson, "The Unix Time-Sharing System," *The Bell System Technical Journal*. July-August 1978, vol. 57, no. 6, part 2.

[19] K. J. Sullivan and D. Notkin, "Reconciling Environment Integration and Component Independence," *Proceedings of the ACM SIGSOFT / SIGPLAN Software Engineering Symposium on Practical Software Development Environments*. December 1990, vol. 15, no. 6, pp. 22-33.

[20] M. Tedjini, I. Thomas, G. Benoliel, and F. Gallo, "A Query Service for a Software Engineering Database System," *Proceedings of the ACM SIGSOFT / SIGPLAN—Software Engineering Symposium on Practical Software Development Environments*. December 1990, vol. 15, no. 6, pp. 238-248.

[21] I. Thomas, "PCTE Interfaces: Supporting Tools in Software Engineering Environments," *IEEE Software*. November 1989, pp. 15-23.