HEWLETT PACKARD

Adding Control Integration to PCTE

Huw Oliver
Information Management Laboratory
HP Laboratories Bristol
HPL-91-62
May, 1991

CASE, software engineering environments, PCTE, SoftBench

The PCTE interfaces provide data-integration services. In a good Software Engineering Environment (SEE), however, it is also necessary to have control integration to automatically start tools and share services. We report on our intermediate practical experience of adding control integration to PCTE. More precisely, we show how Broadcast Message Services can be layered on the PCTE platform, thus forming a SEE framework that spans the tool integration dimensions.

1 Introduction

The computer support environment provided for software engineering today typically consists of a set of standalone tools. These tools are monolithic. These tools do not usually cooperate. They cannot access each other's functionality. They have no access to each other's data (and would not be able to understand it if they could). Their user-interfaces differ widely.

The tools are monolithic in that they provide many of the services more naturally provided by the framework within which they operate or by other tools. For instance, some document processing tools today offer version control even though it is also provided by configuration management tools. Such tools provide so much because the tool providers have no way of composing tools from small modular pieces.

We are interested in how the framework can provide different types of composition or integration services. These integrating services would help tools to be smaller, more modular and built into the support environment as needed by the software engineer.

A complete support environment for software engineering will be a large, complex system. Neither the high level of financial resources nor the wide range of expertise required to provide all the elements of a support environment will be found within a single organisation. The use of open standards for these elements is an essential enabling factor for the production of quality SEE implementations.

We have looked at two technologies which provide elements of a support environment and investigated how they might be combined. The technologies are SoftBench [2] [4] and PCTE [7] [8] [9].

SoftBench is a product of Hewlett-Packard. It consists of an integration framework and an integrated set of tools.

PCTE stands for "a basis for a Portable Common Tools Environment". PCTE defines an interface to support CASE tools and development environments. PCTE itself does not provide any tools: it is a framework on which to build and integrate tools. The development of the interface has culminated in the ECMA PCTE abstract specification [9] which the ECMA general assembly adopted as an ECMA standard in December 1990.

We have undertaken a prototyping activity to show how these components can be combined. The goals of our prototyping activity are to investigate how to construct a support environment, to learn how to use it and to examine the benefits of working with it. We are using an implementation of version 1.5 [7] of the PCTE interface in our prototyping activities. We report here our intermediate technical results from constructing the prototype SEE framework in the HP research laboratories.

2 Integration Services in an SEE Framework

The ECMA CASE environment framework reference model [1] identifies and defines integration services that a framework may provide to support a SEE, and groups related integration services together. Figure 1 shows the overall structure of the reference model (this is an conceptual architecture not an implementation architecture).

The reference model (RM) can be used to categorise the services offered by an SEE framework. Although the ECMA RM activity was spawned from the ECMA PCTE Standards committee, the RM is completely independent of PCTE. The RM can be used to position standards proposals and commercial products, and helps to understand the relationships between different framework offerings.

This section quickly sketches the services required of an SEE framework in terms of those detailed in the RM.

The RM identifies three main aspects of tool integration:

- Data Integration (addressed by the data repository plus data integration services) is the sharing of data and descriptions of that data (schemas) between the users and tools of the support environment.
- Control Integration (addressed by the task management plus the message services) is the management of cooperation between independently developed tools to achieve a coordinated effect.
- User Interface Integration (addressed by the user interface services) is a common look and feel for tools.

2.1 Data Integration

The maintenance, management, and naming of data entities or objects and the relationships among them is the general purpose of the data repository services. Basic support for process execution and control is also addressed here along with a location service to support physical distribution of data and processes.

The data integration services enhance the data repository services by providing higher-level semantics and operations with which to handle the data stored in the repository.

2.2 Control Integration

A high level of control integration implies that a tool can invoke or stimulate another tool to perform some piece of the software process. Control integration is governed by the extent

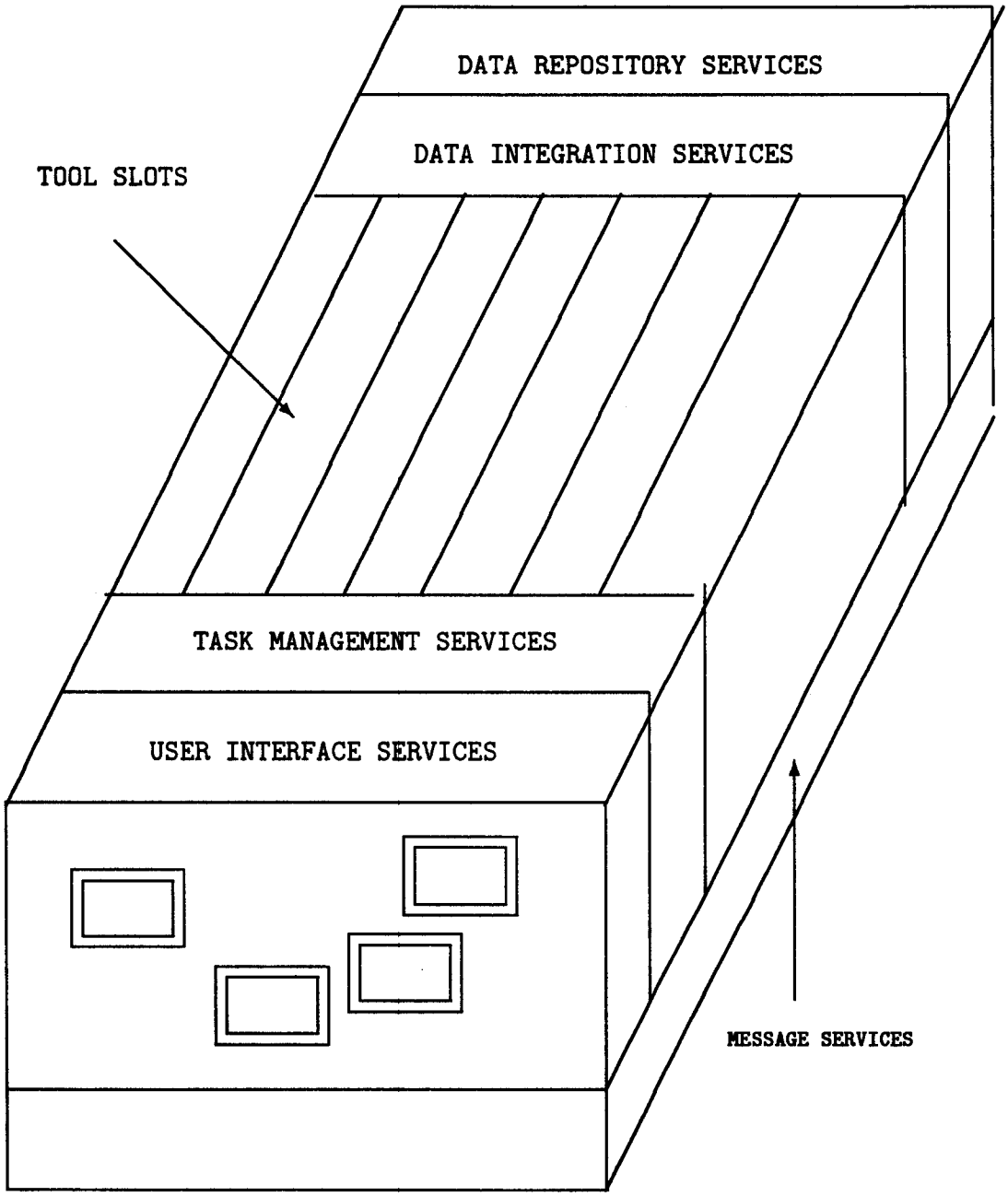

Figure 1: Reference Model Structure

to which a tool makes it possible for other tools to invoke the functionality it provides, and the extent to which the tool calls other tools to communicate changed circumstances.

The message services aim to provide a standard communication service that can be used for inter-tool and inter-service communication.

2.3 User Interface Integration

User interface services are required by all applications. Efforts such as OSF/Motif provide generic services which are suitable for SEEs.

3 Enabling Technologies

The application of the RM to an interface definition will result in a detailed analysis of what SEE framework services are covered by that interface. We have carried out several such applications. Included among these are the application of the RM to PCTE and to SoftBench.

The following important points result from positioning PCTE 1.5 and the tool integration component of HP's SoftBench environment against the RM:

- PCTE covers the majority of the data integration facilities;
- SoftBench addresses control integration via its Broadcast Message Server.
- SoftBench addresses user interface integration via OSF/Motif.

SoftBench treats control integration as an orthogonal issue to data management. SoftBench can be used with many different repositories. We chose PCTE because of its wide coverage of data management facilities and because it is a standard tool portability platform.

From the point of view of integration technology, SoftBench and PCTE are complementary and add value to one another. This analysis encouraged us to investigate the combination of the SoftBench and PCTE integration technologies in practice. We next give an overview of each of SoftBench and PCTE and then describe our approach to combining them.

3.1 SoftBench

The SoftBench environment consists of a set of integration services and an extensible set of tools that communicate by sending and receiving messages. From the point of view of an environment builder, SoftBench consists of the Broadcast Message Server(BMS), the Execution Manager (EM), the user interface, support for distribution, the set of tools and the Encapsulator. The BMS and the Execution Manager are described in further detail by Cagan [2]. Further information about SoftBench tools can be found in Gerety's description [4].

3.1.1 SoftBench Integration Services

1. SoftBench's Broadcast Message Server (BMS) enables executing SoftBench tools to cooperate in supporting a software engineer to carry out tasks. Executing tools in SoftBench send a message to the BMS when they: require a service; have performed an action that may be of importance to others; or have a failure to report. The BMS forwards this message to all the executing tools that have registered interest in a "message-pattern" that the message matches (so the message 'broadcast' is in fact selective). Messages can be sent to the BMS by tools so they can register and unregister interest in patterns.

2. The Execution Manager (EM) in SoftBench keeps track of the tools that are executing. The execution manager cooperates closely with the BMS so that when a request message is received by the BMS, the EM determines whether a new tool should be started to service that request or whether the request can be satisfactorally handled by a tool that is already running. SoftBench tools are grouped into classes. Differing criteria can be applied for differing classes of tools. A class is a set of tools that provide equivalent services. Example tool classes are EDIT, COMPILE, or DEBUG.

3. All SoftBench tools have a common look and feel which conforms to the OSF/Motif [3] standard.

4. SoftBench is designed to operate over a distributed network of workstations, and offers distributed computing support of three kinds. Firstly, SoftBench can start tools and support transparent communications between tools executing on remots hosts. Secondly, SoftBench tools are built on the network transparent X Window System which means that programs can run on one system and display visually on another. Thirdly, SoftBench supports access to remote data.

3.1.2 SoftBench Tools

The initial set of tools delivered with the SoftBench product concentrates on support for developing, versioning, and debugging C and C++ programs. An increasing number of Encapsulated tools are available to extend the core environment, for example tools for configuration management, documentation, structured analysis and structured design, and testing.

Some fundamental SoftBench tools of particular relevance to our work to date are the Tool Manager, the Message Monitor and the Development Manager.

- The Tool Manager presents a way for a user to directly invoke tools. While this is useful at the start of a work session the user will later take advantage of the BMS and EM support for control integration. The user will normally be working within a particular tool (such as a debugger) and will be accessing the functionality of other tools from within that tool.

- The Message Monitor displays all messages that get sent in the environment.
- The Development Manager offers a view of the underlying file system, including an indication of the type of information held within the file (e.g. C source or build information). It also presents a set of operations available on those files (such as versioning). The set of operations made available dynamically matches the type of the file (e.g. it is not possible to even try to check-out a non-versioned file).

We see in section 4.2.3 how we have modified these tools to run on a combined SoftBench and PCTE framework.

The user sees tools working synchronously because cooperation between tools can be specified and the SoftBench system supports the execution of that cooperation. For example, should the user change the source code of a program while working in the static analysis tool, notification of those changes are automatically forwarded to any editor working on the source file for that code. The SoftBench user is also presented with seamless functionality (synergy) in that the services provided by one tool appear (to the user) to be available from several other tools also. For example, code can be recompiled through a user request to the debugger (which is automatically forwarded to the build tool via the BMS). The real benefit of SoftBench to a software developer is that it makes available these advantages of well-presented control integration.

SoftBench provides a further tool called the Encapsulator. This tool enables existing tools to be integrated into the SoftBench support environment without source code modification. It enables a wrapper to be developed for a tool so that its input and output is monitored. Suitable SoftBench messages can then be sent and acted upon by the encapsulated tool, and a SoftBench user interface can be developed so that the tool looks as well as behaves like a true SoftBench citizen (although this holds for a particular set of tools: those that can use standard input/standard output and that can be decoupled from any bitmapped screen handling they do).

3.2 PCTE Integration Services

A major contribution of PCTE is its Object Management System (OMS), designed to meet the data integration needs of CASE tools. The OMS provides the ability to model relationships between data objects, by supporting a variant of the entity-relationship-attribute data model. Object management facilities include typing, schemas and transactions to support data structuring and data sharing, and to maintain data integrity.

PCTE provides a complete interface for the tool writer, including process management and inter-process communication. PCTE provides synchronous and asynchronous calling of processes on local or remote hosts. The services provided are at a higher level of abstraction than those typically provided by the operating system. PCTE inter-process communication

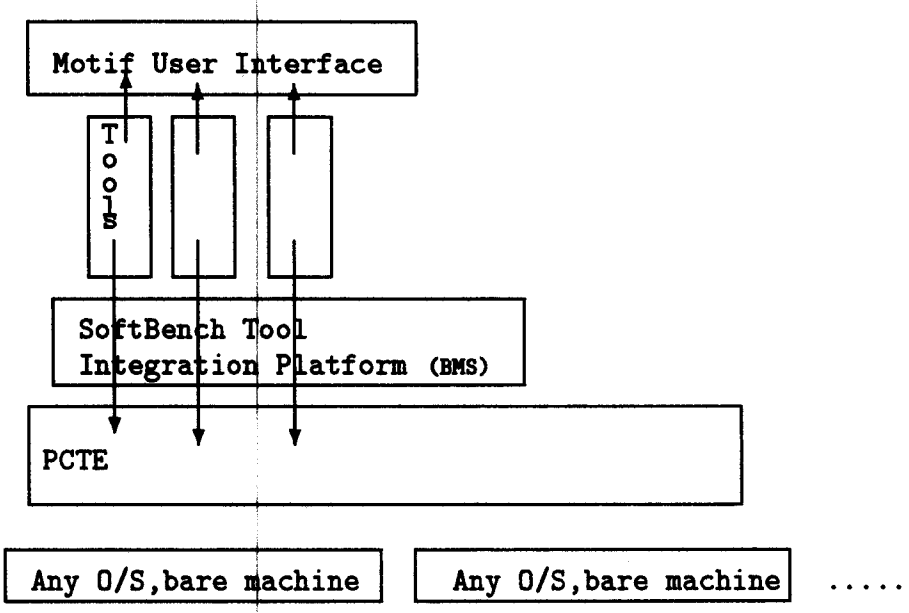

Figure 2: Prototype Architecture

services are provided via the PCTE message queue. These services are closely modelled on the X/Open System V UNIX[1] interfaces. These services are also at a higher level of abstraction than, say, socket based communication primitives.

The hardware architecture for a PCTE system is a network of bitmapped workstations connected by a high speed reliable LAN. PCTE is a distributed architecture, and all the object management and process management facilities are transparently distributed.

4 Prototyping Experience

In this section, we report on our intermediate results from building a prototype SEE framework.

4.1 Architecture

The architecture of the prototype is shown in figure 2. Because PCTE provides a complete interface for the tool writer and because the BMS control integration serives are at a higher level than the PCTE facilities, we have re-implemented the BMS on top of PCTE.

Each of the boxes shows one of the existing components from which we constructed the prototype. The arrows from the tools show which services were accessed by the tools. Thus the tools are linked in with and make calls to:

[1]UNIX is a registered trademark of UNIX System Laboratories Inc.

1. the Motif X Window libraries;
2. the BMS component of the SoftBench libraries;
3. the PCTE libraries.

It can be seen from the architecture that the SoftBench Tool Integration Platform only provides the BMS services. We are investigating extending this so that tools only access the PCTE services through this intermediate layer. This has the advantages of protecting the system from changes in successive PCTE versions and minimising the task of providing data integration in some way other than through the PCTE object base. It would also mean that existing SoftBench tools could be ported with a minimum of effort to the combined SoftBench/PCTE framework.

4.2 Description of the prototype

We are using the GIE Emeraude implementation of the PCTE 1.5 specifications known as Emeraude v12. It is a complete implementation of the PCTE 1.5 interfaces with additional Common Services (e.g. Metabase, Version Management).

PCTE's claim to provide a portability platform was verified by us when we ported several thousand lines of source code between workstations of different hardware from different manufacturers.

Figure 3 shows some of the elements of the prototype. The top box represents the BMS; the boxes in the second row represent class managers; the boxes in the bottom row represent instances of tools. All communication is via the BMS. We now describe these elements in more detail.

4.2.1 The BMS

The BMS runs as an PCTE process. The BMS communicates with all the tools through the PCTE inter-process communication mechanism of message queues. These replace the socket connections in the SoftBench BMS.

The BMS has an associated message queue whose whereabouts in the object base must be known by all tools. The message queue's location was (arbitrarily) chosen to be linked to the static context of the BMS (static context is the PCTE term for 'program'). Essentially the BMS maintains a 'pattern map' which is a map from tool identifiers to the set of message-patterns in which those tools have registered interest. It continuously reads from its message queue, suspending execution until a message arrives. The message will be forwarded to any interested tools or may cause the pattern map to be updated.

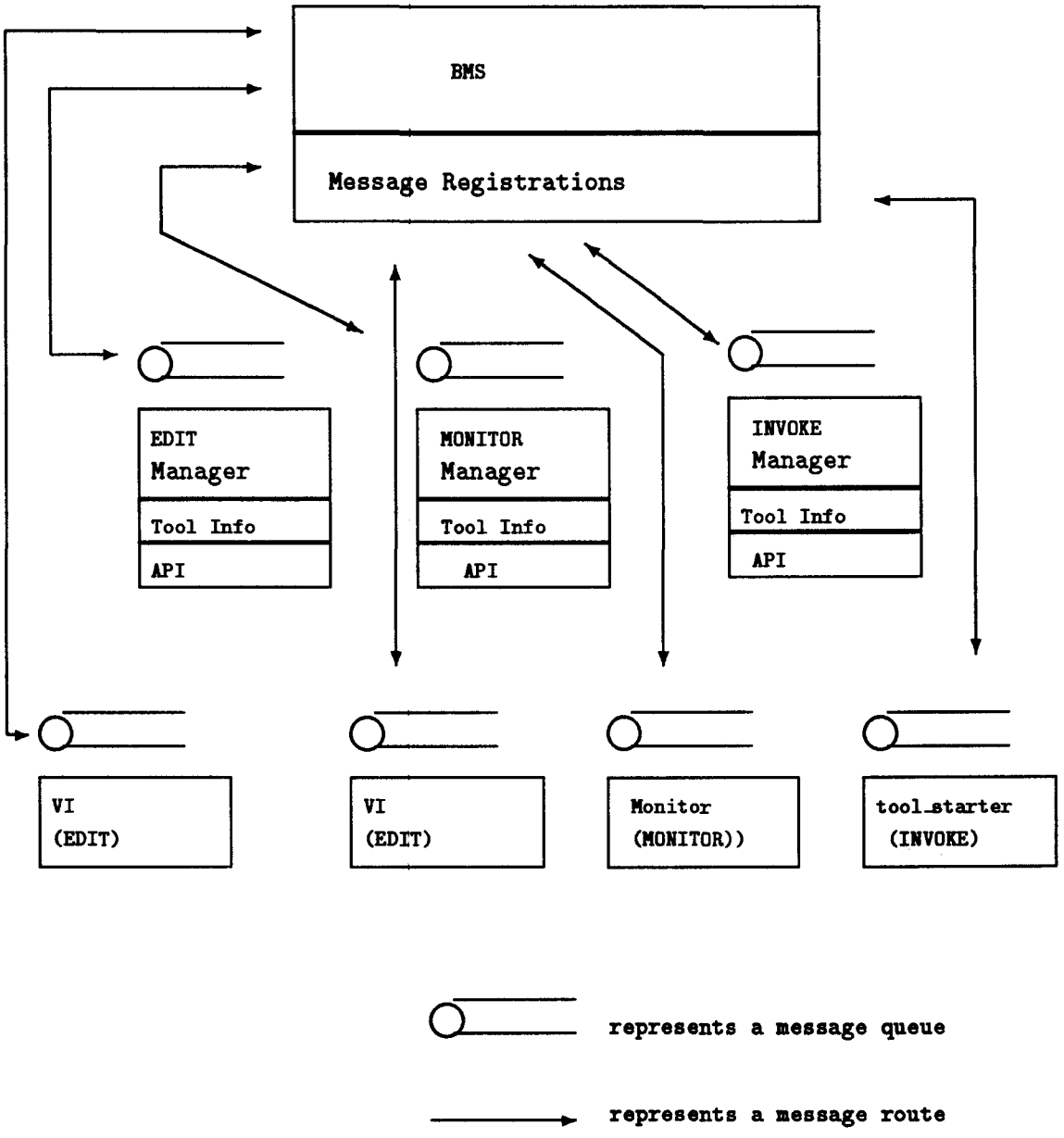

Figure 3: Current Prototype

4.2.2 The Class Managers

Every tool belongs to a class. Each class defines the functionality which tools of that class will provide to other tools. This functionality is accessed by sending request messages to the tool. There is a class manager for each class. The class manager maintains a list of the running tools of its class and carries knowledge of whether there is a tool able to service any given request or whether a new tool needs to be started.

Each class manager runs as a PCTE process. They each have an associated message queue linked to their static context. Each class manager continuously reads from its message queue, suspending execution until a message arrives. Any request message will be forwarded to whichever tool is able to service it. All class managers are very similar except for the knowledge about when new tools should be started up. This knowledge is more complicated in PCTE where objects do not have unique pathnames and where the context of a tool includes the working schema of that tool.

The amalgamation of all the class managers corresponds to the Execution Manager in SoftBench. By separating out the class manager processes we were able to make the decisions about whether to start new tools to handle requests specific to the class of the tool. In SoftBench the Execution Manager used the UNIX execution primitives. In our prototype these have been replaced with the PCTE execution primitives.

4.2.3 Tools

A number of simple tools have been put together for this prototype:

- The *INVOKE* tool corresponds to the SoftBench Tool Manager. It allows the user to select, start and stop tools of any of the available tool classes.
- The *MONITOR* tool corresponds to the SoftBench Message Monitor. It registers an interest in all kinds of messages and displays them. It provides a window onto the BMS activities.
- The *DM* tool corresponds to the SoftBench Development Manager. While the SoftBench development manager gives an interface to the UNIX file system, the DM tool gives a similar interface to the PCTE object base. This enables us to navigate around the object base. The tool includes some version management facilities using the common services provided with the Emeraude product.
- the DMGRAPH tool is a graphical interface tool to the PCTE object base. It navigates the object base via mouse selection of objects, displays the object graph to a user chosen depth, reorientates and manipulates the graphical representation and dynamically manipulates working schemas to provide views on the object base.

- The *EDIT* tool is for editing the contents of objects.

Each tool, like the class managers, runs as a PCTE process. They each have an associated message queue linked to their static context. Each tool continuously reads from its message queue (not suspending execution) until a message arrives. Any request message will be serviced in a tool specific way.

4.2.4 Additional PCTE features of interest

A PCTE installation will typically be distributed over a set of workstations connected by a local area network. The transparent distribution facilities provided by PCTE meant that we did not have to concern ourselves with distribution when designing the prototype. We believe that the SoftBench distribution facilities can be provided on top of PCTE with the added advantage of location transparent access to data.

ECMA PCTE implementations will provide more services than PCTE 1.5. One such service is the ability to respond to events such as access to particular objects in the object base. Adding such services to our existing control integrations services are of much interest and will provide further research directions.

We have not used the PCTE support for concurrency and integrity control and activities. We have not heavily used the schema management facilities.

5 Summary of What We have Learnt

Several points came out of our construction work with respect to PCTE:

- we found the PCTE interface useful in building the BMS and the prototype tools. It provided all the facilities we needed and many of the services were at a higher level than that provided by the operating system.
- PCTE is an effective portability platform;
- we found object identification somewhat confusing, having to switch between pathnames, internal references, external references and volume number, object number pairs. A clear notion of object surrogate would have simplified our task.
- documentation is needed to guide the tool writer through the many design decisions he needs to make. This should include a guide for data integration (how to use the schemas provided and how to write new ones, etc.), and a guide for control integration (how to use the interfaces exported by existing tools and what new message interface a tool should provide, etc.) ;

- a clear and well documented migration path from existing toolsets will be needed;
- The distribution facilities provided by PCTE meant that we did not have to concern ourselves with distribution issues when designing the prototype.

There are many software architecture decisions which should be made by tool writers even if PCTE is not used as the basis for the support environment (such as the production of appropriate schemas and the use of integrating service libraries which hide the underlying technology). These are generally good engineering practices but will protect investment in tools and will ease the transition to PCTE.

The prototyping work at HP Laboratories has proven the feasibility of adding a BMS to PCTE. We are starting a new prototyping phase to experiment with rehosting the SoftBench environment on PCTE.

6 References

[1] A. Earl. A Reference Model for Computer Assisted Software Engineering Environment Frameworks. Technical Report ECMA/TR/90/55, ECMA, 17 August 1990.

[2] M. R. Cagan. The HP SoftBench Environment: An Architecture for a New Generation of Software Tools. *Hewlett-Packard Journal*, 41(3):36–47, June 1990.

[3] A. O. Deininger and C. V. Fernandez. Making Computer Behaviour Consistent: The OSF/Motif Graphical User Interface. *Hewlett-Packard Journal*, 41(3):6–26, June 1990.

[4] C. Gerety. A New Generation of Software Development Tools. *Hewlett-Packard Journal*, 41(3):48–58, June 1990.

[5] F.Gallo G.Boudier and I.Thomas. Overview of PCTE and PCTE+. *ACM SIGPLAN Notices*, 24(2), February 1989.

[6] I.Thomas. PCTE Interfaces: Supporting Tools in Software Engineering Environments. *IEEE Software*, November 1989.

[7] Brussels Commission of the European Communities. PCTE Functional Specifications, Version 1.5, November 1988.

[8] Software Sciences Ltd. GIE Emeraude, Selenia Industrie Elettroniche Associate. PCTE+ Functional Specification, Issue 3, October 1988.

[9] Portable Common Tool Environment - Abstract Specification. Technical Report ECMA Standard 149, ECMA, Geneva, December 1990.