TIGHTLY CONTROLLED PROJECT SUPPORT ENVIRONMENTS

K. Jackson

Rapporteur: Mr. A.W. Brown

Tightly Controlled IPSEs.

Ken Jackson
Systems Designers plc
Camberley
England

1. Introduction

In this presentation I will firstly address the questions of why Integrated Project Support Environments (IPSEs) are considered to be desirable and what constitues a good IPSE. Then I will present two practical examples developed by Systems designers and finally I will attempt to identify some underiying principles of tightly controlled IPSEs.

2. Why are IPSEs required?

There are two fundamental reasons why IPSEs are essential. The first reason is concerned with the "headfull" problem: it is impossible to develop large systems by a single individual, or even by a small team. Therefore we need an IPSE to record and control the interactions between members of the team. The second reason is that it is impossible for a system to be developed from cradle to grave using just a single tool. Therefore we require an IPSE to record and control interactions between the various tools required throughout the project lifecycle. In other words the tools need to be **integrated.**

Thus it can be seen that a fundamental requirement of an IPSE is to enable a team of people to work together in a harmonious and cooperative manner. They must be able to work independently of each other as far as possible, but it is essential that thay cooperate within a management framework. In such a framework it is essential that each individual knows exactly what he/she is expected to do and that the manager can find out how well each individual is progressing with his allotted task. Finally, by integrating the actual progress made by each indiviual, it is possible to calculate a measure of overall progress. Note that by providing facilities for progress to be measured by examining a database, it is possible to obtain a much more objective measure or progress and so avoid such subjective opinions as "90% complete".

A further important aspect of teamwork is the need to check that the project defined quality assurance procedures have been followed. Again, such inspections can be performed via database interrogation and these will show exactly what has been done. Once an object has been developed and tested, and has passed all the appropriate quality assurance reviews, it is possible to place it under change control without any additional operations such as copying . This then means that an object can not be changed again unless such a change is properly authorised by the responsible person.

3. What makes a good IPSE?

Some people believe that the essential property of a good IPSE is that it should nurture **creativity** by allowing the developer freedom, flexibility and extensibility. Other people believe that the essential property of a good IPSE is that it should nurture **orderliness** by introducing control, discipline and accountability. It could be argued that these apparently opposing views represent the respective views of academia and industry. Alternatively, they could be considered to be the respective views of developers and managers.

It is my view that these two different views are not opposing views, but are simultaneous requirements for an IPSE. It is essential that an IPSE offers the least possible hindrance to developers whilst at the same time providing a sound and controlled management capability. Another way of viewing this is that it is essential to recognise that software development is a serious business which must be performed in a disciplined manner by cooperating individuals. Once this basic tenet has been accepted, then the purpose of an IPSE becomes much clearer: it is there to provide facilities to help each member of the team - whether as developer or manager or QA controller - but it must also provide a level of policing which can catch people who transgress the rules - whether by accident or by maliciuos intent. The key is to get the level of policing right so that law abiding members of the community do not feel oppressed, but unlawful members are apprehended.

4. Two example IPSEs.

Systems Designers has been involved in the development of IPSEs for almost 10 years now. Figure 1 shows each of the IPSEs which have been developed, and shows how they are related to each other. Context was our first support environment and it provided an integrated set of tools to support the development of software using the MASCOT [1] design method, the programming language Coral66 [2] and the host-target development technique. Context was developed in the late 1970's to run on Digital PDP11 hosts and most of the then available 16 bit microprocessor targets It is now hosted on the VAX range of computers and has been (and still is) in very widespread use for the development of embedded real-time systems for defence.

Although Context provides an integrated set of tools, its database supports only a single user at a time and can only hold a single version of any object at a time. Perspective was developed in the early 1980's specifically to address these perceived weaknesses of Context. It was also aimed at a much wider market area than UK defence Consequently Perspective supports the programming language Pascal. Perspective is my first example IPSE and I will say more about it shortly.

Perspective is currently in use on over 50 sites throughout Europe and has been successfully used for the developement of the avionics software for the new aeroplane known is the Experimental Aircraft Programme (EAP) built by British Aerospace and which was flying at the Farnborough Air Show during the week of the Newcastle conference Experience with the use of Perspective has shown that its basic concepts are right, but that, inevitably, improvements can be made. Some of the improvements are in the area of team facilities and better policing, but the most fundamental improvement was to make it into an "open" environment i.e to allow users the possibility of adding additional tools. The resulting product is called Perspective Kernel. This indicates that it is a development from Perspective and that it provides a basic set of facilities which form the kernel of any bespoke IPSE developed with it. Perspective Kernel, or PK as we usually refer to it, is my second example IPSE.

Figure 1 shows two environments being developed using PK. Horizon is a new environment to support MASCOT3 [3] and Coral66, whilst Perspective/Ada is an APSE which incorporates the design language discipline of Perspective. The other three boxes on figure 1 all refer to Aspect. This is a collaborative project involving both industrial and academic partners and is partly funded by the UK Department of Industry under their Alvey inititiative. The industrial partners are Systems Designers, International Computers Limited and the Microcessor Applications Research Institute. The academic partners are the Universities of Newcastle on Tyne and York. Peter Henderson describes some of the results of the Aspect research project in another section of this report.

4.1 Perspective

The talk explained briefly the motivation behind the development of Perspective. Perspective provides a design language as a set of extensions to ISO Pascal, which provides a formalism to represent the structure of the real-time embedded system being developed. This structure can be checked and recorded in Perspective's database, and then used to control the implementation team so that no deviations from the designer's intention are allowed. This forces the implementation team to either keep to the designer's intented structure, or to report any problems and/or suggested "improvements" **to the designer.** If a modification is considered necessary by the designer, then he can make the appropriate changes in the full knowledge of the overall design concept and issue a new version of the design to the imlementation team. Thus the designer retains control of the design baseline.

Perspective also provides a development method to help in the technical management of the software development. This is based upon the notion of identifying "milestones" along the development path. These milestones correspond to one or a group of database item versions achieving a particular status. These achievements are visible within the database and can be detected by running standard queries. Thus the manager can obtain, from the database, a record of actual progress achieved. This will ignore any progress towards a milestone and only count actual milestones achieved. Thus a rather pessimistic view of progress will be obtained by this method, but this is considered to be a good idea - especially as it avoids the well known "90% complete" syndrome.

For more details on Perspective please refer to the Technical Overview (annex A of this section) and the slides used during the talk.

4.2 Perspective Kernel

PK provides a foundation layer, which is fundamental to the support of openness, and set of project control facilities built upon the foundation. The foundation consists primarily of an ERA database enhanced with facilities derived from Perspective to support teamworking, version management and configuration control. The project control facilities take the Perspective notion of domains, with controlled sharing of item version between them, and adds further facilities to support the notion that each domain corresponds to one or more "roles" within the development team.

These roles incude the ability to act as a team leader (and delegate responsibility for part of the development to other domains), or to act as a QA reviewer (and either accept and place an item version under change control, or reject an item version because the appropriate procedures have not been followed), or to act as development control manager (and authorise further development of item versions previously placed under change control), etc.

For more details of PK please refer to the PK paper (annex B of this section) and the slides used during the talk.

5. Conclusion.

In the early part of my presentation and in the two examples, I have stressed the importance of teamwork and the fundamental requirement to support teamwork in IPSEs. In this concluding section I have identified three principles which need to be observed in any IPSE which purports to support teamwork.

The first principle is the "need to know" principle. This is related to the equivalent security principle, but in terms of IPSEs it can be interpreted more liberally, because the consequences of a person knowing more than is actually necessary are not serious. In the IPSE world the need to know principle means that each team member has a right to see only his own objects. Objects belonging to other domains are normally invisible unless specifically made visible by that domain. This principle must be interpreted so that each team member can see the objects required for the task in hand. Perhaps more importantly, the team member must not be able to see the private objects which other team members are currently developing - unless or until that team member decides to make them visible.

This leads to the second principle, the "dependability" principle. This states that when an object, such as an item version, is visible to a team member, and that team member has elected to depend upon it, then the object will neither change nor disappear.

The third and final principle is the "permission to do" principle. This principle dictates that a team member is prohibited from doing anything unless it has been specifically authorised. This is the means by which domains in the database can be set up to correspond with a particular team role. It is also the mechanism by which tasks can be delegated by one team member to another.

6. References.

1. JIMCOM, The official Handbook of MASCOT, version 2.2. Available from RSRE, St., Andrews Road, Malvern, Worcs., WR14 3PS.
2. Bond, S.G., and Woodward, P.M., The official definiton of Coral66, Published by HMSO.
3. JIMCOM, The official definition of Mascot3, version 3.1. Available from RSRE, St., Andrews Road, Malvern, Worcs., WR14 3PS.

SYSTEMS DESIGNERS LIMITED

DEC VAX-11

PERSPECTIVE

TECHNICAL OVERVIEW

Systems Designers Limited,
Systems House,
1 Pembroke Broadway,
Camberley,
Surrey. GU15 3XH

Telephone: Camberley (0276) 62244
Telex : 858280 SYSDES G

Q/BC-00/10

Issue 2.0

December 1983

CONTENTS

FIGURES

CHAPTER 1

INTRODUCTION AND MANAGEMENT SUMMARY

PERSPECTIVE is a Programming Support Environment developed by Systems Designers Limited for PASCAL based software systems. PERSPECTIVE provides support for all stages of the software development process from initial design to in-service maintenance.

In particular, PERSPECTIVE is designed to meet the demands of management in controlling project teams, and to provide specialist support for developing and maintaining real-time software for embedded microcomputer systems.

1.1 PROJECT MANAGEMENT

Project Managers are responsible for the timely completion of software systems, within budgets, to specification, and to high quality and reliability standards. This is a demanding task.

- Projects can be technically complex, often involving real-time systems and parallel processing on state of the art hardware.

- Projects can be administratively complex with information on progress and requirements coming from many sources, in many forms, with varying reliability and availability.

- * Projects can be organisationally complex with coordination required through all the development stages of:

  a) design

  b) implementation

  c) quality assurance

  d) in-service maintenance.

  In addition, coordination is usually required between team members within these stages.

- * Project team members can make local decisions without realising their wider implications, which can result in significant reworking to meet the system specification.

- * Projects require specialist resources, both human and material, which are often not available when required, and are always in short supply.

A key element of PERSPECTIVE is the support it gives to Project Managers by addressing all of these problems.

1.2 EMBEDDED SYSTEMS

Embedded systems are microprocessor systems which are incorporated as a component in other devices and whose presence is often not immediately evident. They are frequently used for control purposes and operate continuously in real time.

Embedded systems have a number of characteristics which add to the complexity of the implementation task.

- * They tend to have severe timing and memory constraints.

- * They often make use of specialised peripherals which are only available on the operational hardware.

- * The actual operational hardware configuration may not be available until late in the development cycle.

- * The software produced is complex. Frequently it has many parallel paths of execution and extensive interaction with its environment.

- * Embedded computers are totally unsuitable for software development. They do not usually support an operating system or any of the other facilities required by software development tools. Software development must be carried out on a separate "host" computer.

- * Embedded software must be tested on the real hardware where it can run in real time with real peripherals. Usually this means losing all the diagnostic aids of the host computer, and leaves the programmer with no easy means of testing or debugging the system.

PERSPECTIVE provides a unique solution to the problems of developing and maintaining embedded software. The main features of this solution are:

a) A methodology to handle the design of complex software and to provide a safe execution environment for sequential programs in a concurrent system.

b) A hosted development toolkit which supports program development and can be extended to cover the full life cycle of a product.

c) A facility to load application software into a target computer over a serial link, which also allows debugging of the software running on the target from a host computer terminal.

d) A standard run-time package which can be tailored by the user to meet specific project requirements.

In summary, PERSPECTIVE provides the methodology and the tools to ensure an effective solution to the problems of embedded system software development.

CHAPTER 2

THE PERSPECTIVE METHODOLOGY

PERSPECTIVE supports the development and operational use of computer based systems. The emphasis is on embedded computer systems, particularly those which will use microprocessors for their target hardware, but the approach supported by PERSPECTIVE can also be used for systems development on mainframe and minicomputers.

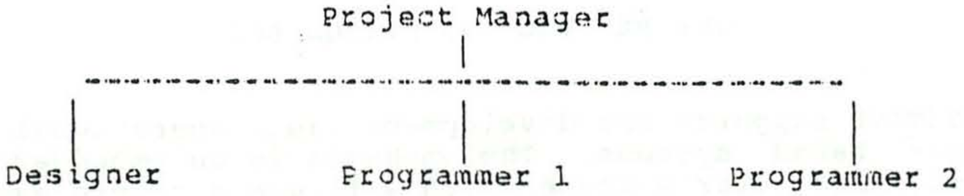
2.1 MANAGEMENT ASPECTS

For all but the simplest of embedded systems it is necessary to set up a team (or teams) of people to produce a system. This immediately raises the management problems of allocating tasks to people or teams and of making sure that the individuals communicate effectively with the rest of the team. It is important to ensure access by an individual to computer resident information as needed, while preventing interference with information being relied upon by others.

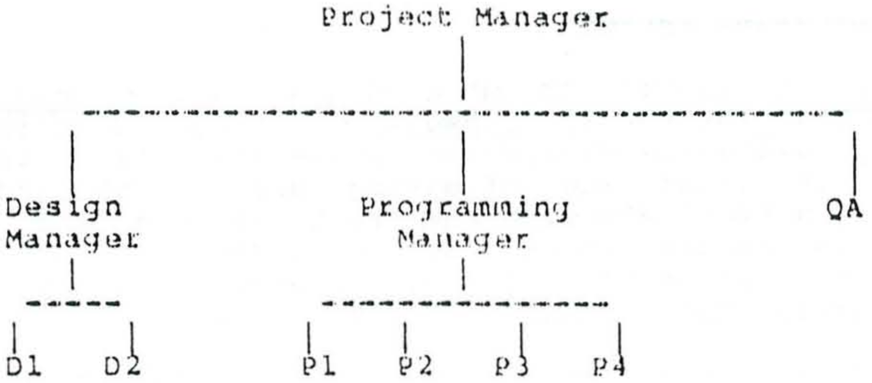
PERSPECTIVE recognises that application systems are produced by teams of programmers and provides facilities to enable the individuals to work independently as far as possible. At the same time it provides a disciplined environment in which programmers can interact with other team members and their software components, and still be confident that, although the components have been developed separately, they can be successfully combined at the appropriate moment. Further, PERSPECTIVE recognises and provides facilities to control the avalanche of version management problems which inevitably occur once a formal issue of a product is made.

In order to achieve this degree of control, PERSPECTIVE maintains a central database of information on all items within the system, including their development status, and the rights of individual team members to use or change them.

Normally a PERSPECTIVE database is created by the Project Manager to reflect the organisation of the people in that project team. For example a small project database might have the structure:



A more complex one might be:



Each team member has a personal view of the database but is prevented from accessing the sections of the database controlled by other team members unless explicit permission has been given.

During coding and testing the programmers will be developing and correcting their own components, and also making use of components produced by other members of the team. PERSPECTIVE will allocate a version number to each new instance of a component, and the compilation process will automatically use the latest version of each component unless particular versions have been specified.

The PERSPECTIVE discipline extends to prohibiting team members from changing components produced by their colleagues. If there is a requirement for this it must be raised with the project manager who can assess the requirement, and its impact, and decide accordingly.

At any time during the project the manager can interrogate the PERSPECTIVE database to determine the precise state of progress on any part of the system. By simple database queries confirmation can be obtained, for example, that the following milestones have been achieved.

- * Item has been created during overall design.
- * Item has been checked for consistency with other items in the overall design.
- * Algorithmic content of item has been written, and checked by the compiler for syntactic and semantic correctness.
- * Item has been tested locally and found to be sufficiently sound to be incorporated into a test configuration for the real target.
- * Item has been compiled for a real target processor.
- * Item has been sufficiently proved on the target to warrant being handed over to the QA team for more thorough testing and certification.
- * QA are satisfied with the component and have made it available to a wide community of users.

Monitoring progress against events such as those in this list can be performed at any time and at any level within the system. The PERSPECTIVE approach ensures that the Project Manager's view of progress in any area represents real achievement.

The version control provided by PERSPECTIVE during system development also gives the Project Manager an opportunity to identify and tackle problem areas at an early stage. If, for example, a particular module or subsystem is undergoing many changes in the compilation or testing stages, then there is clearly a need to review these areas in detail. Conversely, reviews of stable elements of the system can be reduced or even omitted.

All of these features stem from the central database within which all elements of the system are stored, and through which the various tools and commands of PERSPECTIVE control access and actions, and record progress. All of this information is available to the Project Manager throughout the life of the project.

In this way, PERSPECTIVE provides the support which allows Project Managers to spend their time productively on real issues and to help ensure that their team produces systems on time, within budgets, to specification, and to the highest quality and reliability standards.

2.2 OVERALL DESIGN

The starting point of the PERSPECTIVE methodology is the existence of a requirement. However it should be expected that the design process, particularly the earlier phases, will lead to a clarification of the requirement and to the exposure of areas where trade-offs between design and requirement may be made.

The first stage of a design is to analyse the requirement from a data flow point of view. This will inevitably expose the need to handle events which occur simultaneously. This is represented as a set of inter-communicating "subsystems" using the diagrammatic notation shown in Figure 2.1.

As the analysis proceeds and subsystems and their intercommunication requirements are postulated, each subsystem and each interface must be justified by identifying its purpose and its function. The first stage of this design is complete when a "structured walkthrough" of the system diagram(s) has been completed to the satisfaction of all concerned.

2.3 SUBSYSTEM DESIGN

The next stage of the design is to decompose each subsystem into its components.

In the decomposition of a subsystem, consideration must be given to the method of implementation for each interface. There may also be a need to implement asynchronous functions.

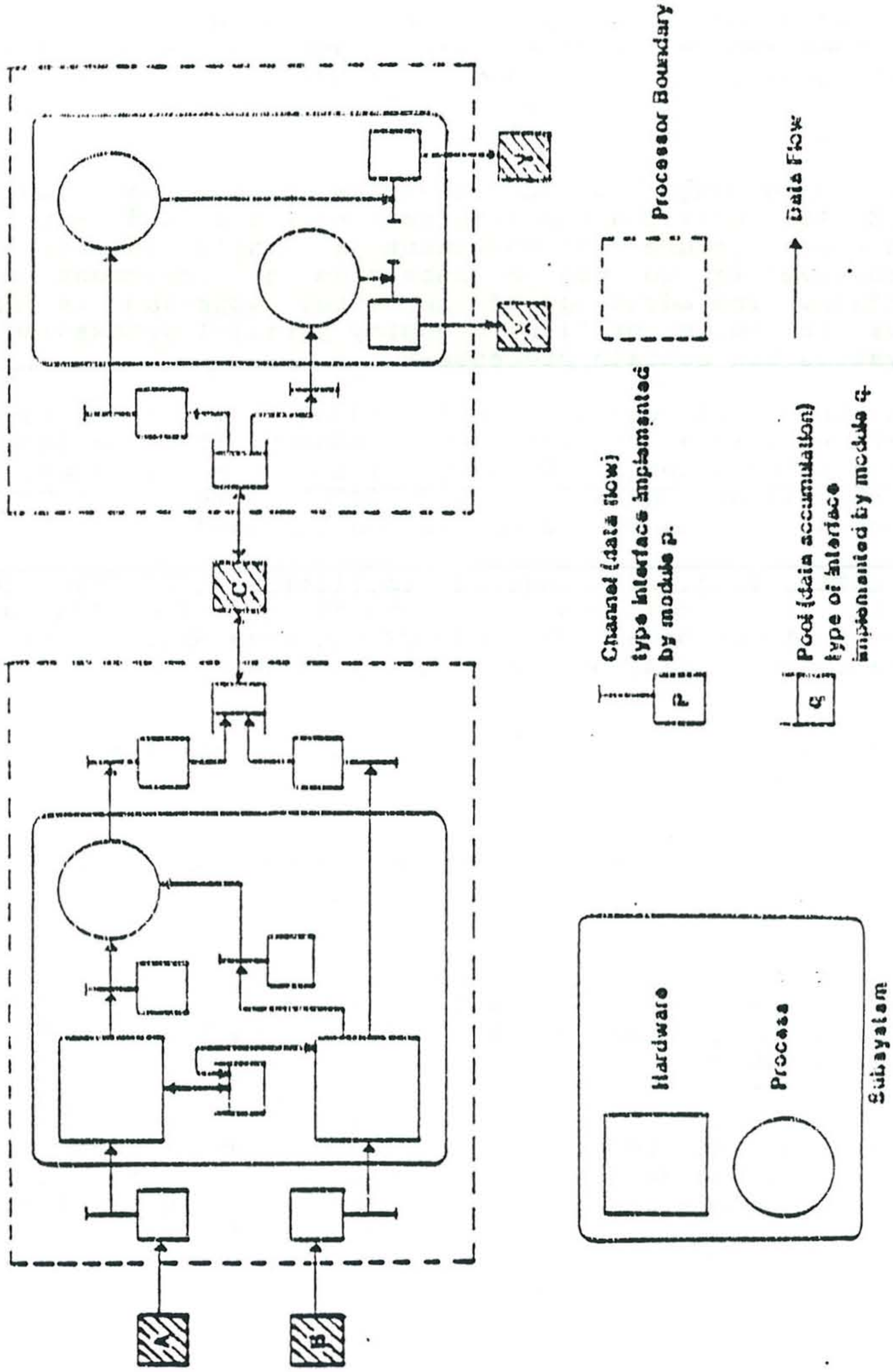

A
B
C
X
Y
Hardware
Process
Subsystem
Processor Boundary
Data Flow
p
Channel (data flow) type interface implemented by module p.
q
Pool (data accumulation) type of interface implemented by module q.

Figure2.1 DIAGRAMMATIC NOTATION

The asynchronous functions can be represented by further subsystems (which in turn will have to be decomposed) or processes which are the fundamental unit of parallel processing in PERSPECTIVE. Processes as well as subsystems can communicate only via interfaces.

When considering how an interface should be implemented PERSPECTIVE provides two alternatives. The fundamental method is to use a module to implement a single interface. The alternative is to use a subsystem to implement a single interface. The advantage of the latter approach is that it allows the implementation to employ parallel processing (since subsystems can contain processes).

The technique of using a module/subsystem to implement an interface allows Parnas' design concept of "data/information hiding" to be adopted, whilst permitting a module/subsystem to use other interfaces allows PERSPECTIVE to represent Dijkstra's concept of "successive levels of abstraction".

PERSPECTIVE provides language facilities (as extensions to PASCAL) for recording these aspects of the formal design within its central database. PERSPECTIVE can then make use of the use of the recorded design in the following ways :-

* Validating the completeness and consistency of the design itself.

* Defining re-usable modules and thus providing the basis for a software component library.

* Identifying and assigning tasks for individual implementors, who will each be given an appropriate view of the design through their section of the database.

* Ensuring that the implementation adheres to the recorded design. PERSPECTIVE will not compile a module which does not conform to the interface design, or for which there is no recorded design. In doing so, it provides a means of prohibiting implementors from making unauthorised local decisions on design issues.

* Providing management information on the impact of proposed design changes.

2.4 IMPLEMENTATION AND TESTING

During coding and testing the programmers will be developing and correcting their own modules, and also making use of components produced by other members of the team. PERSPECTIVE will allocate a version number to each new instance of a module; the compilation process will automatically use the latest version of each module unless particular versions have been specified.

The PERSPECTIVE discipline extends to prohibiting team members from changing components produced by their colleagues. If there is a requirement for this it must be raised with the Project Manager who can assess the requirement, and its impact, and decide accordingly.

Within a module or a process the programming language is standard PASCAL, with a number of extensions to meet the needs of real-time systems, and to allow the modularity concepts of interface, module, process and subsystem to be expressed.

These extensions allow modules to be compiled separately, and go so far as providing for the automatic construction of a complete system. When an element of the system is changed, PERSPECTIVE can automatically determine those modules affected by the change and can rebuild the system with the minimum necessary recompilation.

PASCAL and the real-time extensions provide the user with a language which can be used for virtually all embedded microcomputer applications. This is achieved by restricting the language extensions to low-level hardware control and by not defining a particular real-time regime within the language. Basic facilities such as process synchronisation are built into a standard package written in PASCAL and supplied in source form so the user can make modifications to meet precise needs, or even replace the whole package if required.

PERSPECTIVE includes a wide range of testing and debugging tools, all of which refer to data items by their familiar PASCAL names. These tools can be applied to the initial testing of modules on the host machine, and also to subsequent testing on the target microprocessor under control from PERSPECTIVE on the host.

Down-line loading is supported by connecting the host and target computers with a simple communication link. Debugging is achieved by use of the same link.

CHAPTER 3

USER FACILITIES

3.1 THE PERSPECTIVE DATABASE

The database holds all the source files and any machine code files which have been constructed. In addition it holds all the structural information concerning the relationship between interfaces, modules, processes and subsystems, together with the managerial information relating to version control and user access rights.

Of particular importance in the database is the ability to maintain dependency information. This is used to detect automatically items in the database that have become invalid as a result of user action (e.g. editing) and to re-create the dependent items when they are required.

The database manager within the PERSPECTIVE system therefore has two roles: it provides support for database access by the tools and also provides the checking and coordination of the application system's configuration, keeping track of versions and dependencies.

The user invokes the various facilities by typing the appropriate command. This "user interface" has an overall style to which all commands conform. This interface also provides a "security barrier" between users, and prevents any user from performing operations which are not authorised or which could inconvenience other users.

3.2 PERSPECTIVE COMMANDS

3.2.1 Access To The Database

Relevant commands are:

- * SET USER
- * EXIT
- * SET PASSWORD
- * SET CONTEXT

SET USER is the command which allows a user to start an interactive session with PERSPECTIVE. When PERSPECTIVE is invoked, the user enters the SET USER command to nominate the domain in the user hierarchy in which work is to be done. By quoting the relevant password for access, the user will be allowed all the privileges available in that domain. Thus the user will be able to create entities with unique ownership; read and use owned entities in the database; read and use entities made public by other users.

EXIT is the command which returns the user to the control of the host operating system.

SET PASSWORD allows a user who has logged in to change the password to be used for future entry to the current domain.

SET CONTEXT allows a user to select the target processor and software environment which will be used for all compilations and testing operations which do not explicitly quote a particular target.

3.2.2 Organising The Database

Relevant commands are:

- * CREATE USER
- * CREATE ITEM
- * INSTALL
- * FIX
- * FREE
- * DELETE
- * PURGE

The commands discussed in this section are concerned with the local use of the database by an individual team member excluding the use by that person of items created by other team members. This latter aspect is discussed in Section 3.2.3.

CREATE USER is the command used to create a new user domain in the database.

CREATE ITEM is the command used to create new textual items in the database. It will allow systems, subsystems, interfaces, modules or processes to be created (in any order), but no dependencies between these items are established until the items are INSTALLed. The text of the new item can be copied from a host file, or from an existing PERSPECTIVE item. The EDIT command may be used to modify the copied text.

PERSPECTIVE does not provide a 'MODIFY' command. Instead, the EDIT or CREATE commands are used to create the next version of an item. This philosophy is an important element in the PERSPECTIVE approach to configuration management and version control.

INSTALL is the next command in the sequence and will check that the 'structural' content of the item is consistent with other information already in the database. During this process PERSPECTIVE will attempt to INSTALL any items on which the initial object of the command depends (recursing to any required depth). The result of this action will be that all the inter-item dependencies will be detected and remembered.

The FIX command is used to freeze the versions of the items used in a particular system or component. By default PERSPECTIVE uses the latest versions of items unless FIX has been used to specify otherwise.

The FREE command reverses the effect of FIX.

The DELETE command allows a user to remove an item from the database. When a development version is deleted then all the items dependent upon the deleted item are automatically deleted. A FIXed version may not be deleted unless it has no dependent items.

The PURGE command deletes all but a specified version of an item with the same constraints as DELETE.

3.2.3 Publication

Relevant commands are:

- PUBLISH
- ACQUIRE
- DISPOSE
- WITHDRAW

Publication is the PERSPECTIVE process of making completed modules, subsystems, or systems available to other members of the project team. The item(s) to be made available to the team members must be FIXed before they are published.

Although each team member has the opportunity to use PUBLISHed items, those items which are required for use must first be ACQUIREd. Only the topmost items need be specified since the items on which they depend will be ACQUIREd automatically.

The ACQUIRE command allows each user to control the elements of the system intended for use in the section of the database allocated to that user, and to advise PERSPECTIVE that these items may not be WITHDRAWn by other team members.

The DISPOSE command is used when previously ACQUIREd items are no longer needed by a particular team member.

The WITHDRAW command reverses the effect of PUBLISH except for its impact on users who have previously ACQUIREd the items. WITHDRAW will not be allowed on any items which have been ACQUIREd but not DISPOSEd.

3.2.4 Transferring Items To Other PERSPECTIVE Databases

Relevant commands are:

- EXTRACT
- OBEY

The EXTRACT command enables a set of software components to be copied from the database into a set of files on the host operating system. The resulting files will be in normal sequential text format and will be suitable for use by CREATE to install that set of software into a different PERSPECTIVE database.

In addition to creating the text files, PERSPECTIVE will also generate a file of commands to CREATE these items in the new database. This set of commands can be executed in the new environment by issuing an OBEY command referencing this file.

3.2.5 The Query Facility

The relevant command is :

- SHOW

The SHOW command provides general purpose query facilities enabling the user to determine:

a) which items in the database are being used by a specific item

b) which items in the database are using a specific item

c) the source text of an item, optionally with installation and compilation errors embedded in the listing, if appropriate

d) which items are owned by the current user domain

e) which items have been PUBLISHed

f) which items have been ACQUIREd by the current user domain

g) the names of the user domains which have been CREATEd

h) whether INSTALLation of an item has been attempted and achieved

i) whether compilation of an item has been attempted and achieved, and for which targets.

The reports generated in response to a query command are normally displayed on the user's terminal but can be directed to a PERSPECTIVE database text item, a host system file or the host system printer.

3.2.6 The Log File

Relevant commands are:

- * SET LOGGING ON
- * SET LOGGING OFF

If a permanent record of all or part of any PERSPECTIVE session is required, then a log file may be used to record commands and responses as they appear on the screen.

The SET LOGGING ON command acts as a switch to activate this process, and the SET LOGGING OFF command will de-activate it.

These commands can be used as and when required during a session.

3.3 THE EDITOR

The Editor used by PERSPECTIVE is the standard DEC VAX/VMS Editor, EDT, and the full facilities of this product can be employed to change existing PERSPECTIVE source text, or to introduce new text items. New items are created as Version 1, whilst existing items have their version numbers incremented by one.

3.4 THE COMPILER

The Compiler is a major component of PERSPECTIVE and is responsible for translating the user's source text into object code for the chosen computer type. It deduces which items of software need to be compiled in order to construct the item requested. It then compiles them in the appropriate sequence, invoking all the necessary passes of the Compiler. For interpretive checkout purposes, only the first pass (PASCAL to P-code) is invoked.

3.5 INTERPRETIVE CHECKOUT

Interpretive Checkout provides facilities for full dynamic checkout of a system or program. Dynamic Checkout is usually performed using a test harness and can be used to investigate the dynamic behaviour of a module or subsystem.

The major facilities offered are:

a) Source level display and alteration of PASCAL variables

b) Incremental program execution

c) Breakpoints

d) Trace of program execution

e) Monitoring and display of variables during execution

3.6 TARGET CHECKOUT

Target Checkout allows the user to connect the host computer to on-line to the target and then use the standard PERSPECTIVE debugging facilites available for Interpretive Checkout. Debugging commands entered at the host result in messages being sent up and down the line and the appropriate debugging actions taking place on the target. Most of the intelligence required for this is located on the host so the overhead for the target is small. One of the main features of on-line debugging is that it is equally effective during program development and during later maintenance and upgrade.

The following facilities are offered:

a) Down-line loading into target hardware

b) Control of target execution

c) Source level display and alteration of PASCAL variables

d) Incremental program execution

e) Breakpoints

f) Trace of program execution

g) Monitoring and display of variables during execution

As far as possible the Target Checkout facilities have been made identical to the Interpretive Checkout facilities, and have the same user interface.

3.7 THE PROM FORMATTER

The PROM Formatter, using the output from system construction, is capable of slicing each contiguous block of memory both vertically and horizontally to produce blocks of formatted data for PROM programming equipment.

The facilities offered are:

a) Location of a system at a specified address and checking that the system size is within limits

b) Capability for splitting a system into a number of non-contiguous PROM regions

c) Horizontal and vertical "slicing" of the memory image to produce data for each of a set of PROMs

d) Generation of PROM programmer data in the appropriate format

e) A user specified contingency factor to allow module expansion without relocation

f) Easy reprogramming of PROMs affected by a source code change.

3.8 TARGET SOFTWARE

In addition to the software package on the host computer, PERSPECTIVE is supplied with a target software package which provides host-target protocol and the real-time regime. The target package is supplied in source text form to the customer who is free to use it, change it, or to replace it with a package written to support other programming techniques.

The facilities offered are:

a) Standard host-target protocol for use during development

b) Standard parallel processing regime support

c) Standard environment for ISO/BSI PASCAL.

PERSPECTIVE KERNEL - AN OPEN-ENDED PROJECT SUPPORT ENVIRONMENT

Peter E. Rusling *

ABSTRACT

This paper describes the Perspective Kernel - a new product from Systems Designers which takes and enhances the features of their award-winning Perspective system to provide a general-purpose Integrated Project Support Environment (IPSE) framework.

PK (as it is generally known) is primarily targetted for use on major real-time software projects, which typically require a number of teams of development staff, often geographically separated, to produce very sophisticated and high-quality systems to a strict budget and timescales.

In this scenario, it is most important that effective tools are used to support the most suitable methods and languages, and that the necessary level of control can be applied to the development process. PK supports these requirements by its ability to embrace any tools (and thus any method and language), and to apply its extensive project control facilities uniformly throughout the development lifecycle.

BACKGROUND

Since 1973 Systems Designers has had a major involvement in the design, development and supply of compilers and tools in support of projects developing real time systems.

The Groups' CONTEXT System was the first in the world which offered host/target debugging capability to projects developing systems for embedded micro-computers. CONTEXT was also the first system to provide programming support for a major design methodology - in this case the MASCOT real-time design method. The database for CONTEXT supported a single user, but as more powerful microprocessors became available development teams became larger and the problems of interaction between team members needed to be addressed.

In 1982 Systems Designers introduced the Perspective system to support these larger project teams. With Perspective, team members are able to undertake their individual activities in their own private area of the database, whilst activities requiring collaboration with other team members are controlled at a team level. Other features offered by Perspective included a design language to control the development process, and configuration and version control features to support the development needs of teams as well as of individuals. In total these innovations provide for the management and design control essential to the success of such projects, while still giving strong support to the individual team member performing a particular task.

The development tools supported by Perspective are based around the Pascal programming language. Today's requirement can best be expressed as being

* Systems Designers, Pembroke House, Camberley, Surrey, England

for Perspective Ada(R), or Perspective CORAL, or more generally a Perspective system to support any language and any set of tools.

The Perspective Kernel System has been developed to satisfy this need for an "open" Project Support Environment.

INTRODUCTION

The "open-ness" of PK allows the project manager to choose the most appropriate language and tools for his project and, by embedding these within PK, to create his own project-specific, company-specific (or even product-specific) environment.

This "open-ness" is achieved by a clear separation between the underlying technology required for a support environment, and the particular functionality which a given environment should provide.

Central to PK is a set of facilities which provide low-level services to the project control functions and to the tools which will operate within the PK framework.

These facilities provide such services as data modelling, data management, file handling, terminal handling, and tool management. All of these facilities are made available through a formal interface which is known as the Public Tool Interface (PTI). Tools may be developed to use this interface, and may be integrated together to provide cohesive support to the software development process.

A special part of this PTI enables the use of tools which have no knowledge of PK: this is known as the Open Tool Interface. Use of this capability allows existing tools to be installed in PK, and permits PK to be introduced to control projects which are already underway.

A most important consideration in the design of PK is the need for complete resilience and immediate recovery from catastrophic events. This has been achieved by a transaction-oriented approach which assures database integrity up to the last successfully completed transaction.

PROJECT CONTROL FACILITIES

On major software development projects, the team members must work together, concurrently and co-operatively. Control must be exercised on what each team member should do, what each team member has done, and what overall progress has been achieved. Throughout the development, quality must be ensured by correct development, acceptance, and change control procedures.

PK provides a set of facilities to allow such control to be achieved; these are described in the following sections.

PROJECT TEAMS AND THE DATABASE

All project data resides (or is accessed through) a multi-user multi-access PK database.

The PK database is divided into domains which each represent a user 'role'. A project member may have many roles on a project, but will use a different

(R) Ada is a registered trademark of the U.S. Government, Ada Joint Program Office.

domain for each role. An individual may have one domain for his main task as Designer, say, and a separate domain when he acts as reviewer for project documentation. Thus domains provide a means for controlling concurrency of access and isolate users from multi-access conflicts.

Each domain has associated with it a set of privileges which define the rights which the corresponding user 'role' has for the operation of project control functions and tools. For example a domain used by a Designer of a sub-system might have privileges allowing read access to specification data, and for use of a design tool; an equivalent programmer domain would have read access both to specification and design data, but no access to the design tool.

ITEMS AND VERSIONS

Database objects are visible to users as 'items' which have a unique name within a domain, and are of one of a set of pre-defined item types. These items may actually be the root nodes of more complex data structures containing attributes, entities, and relationships with other items or entities.

In order to support progressive development and the sharing of stable issues of items, PK provides for versions of items to be established and maintained. The chosen version scheme allows for each version to be accorded a level of importance and formality. This is achieved by identifying each version by a Dewey-decimal-like code where the number of levels indicate the degree of (in)formality as in the following example:

1	top-level formal version
1.1	second-level less formal version
1.2	development of 1.1
1.2.1	sub-development of 1.2
1.2.2	development of 1.2.1
1.3	completed sub-development
2	formal version of 1.3

Within a version level, a source text item may be modified using the standard host editor to produce the next version at that level.

PK also supports the parallel development of variants of an item by the establishment of baselines. This is useful where special variants are required for example for performance optimisation or bug fixing.

QUALITY CONTROL

The version numbering scheme described above operates in conjunction with the PK privilege mechanism for any change of version level. In order to 'move' an item version up or down a level it is necessary to use a 'promote' or 'demote' function which can have a degree of privilege and formality associated with it according to the version level.

For example, to promote item version 1.3 to formal version 2 could require a special privilege which might only be granted to the QA Manager. Similarly, to demote a version for further development could require the authority of of a change control 'role' who would formally declare the purpose of the proposed work.

Thus a project may associate particular QA states with different levels and control the degree of authorisation required to change version level. As a consequence the status of an item is always known from its version number.

Thus version 3.5.2.11 might be an untested item (in common with all items having four levels), and item 3.5.3 would be known to have been approved as a tested item by an independent reviewer (someone with authority to change items of this type from level four to level three). In a similar way an item with version number 8 would be known to be fully approved and certified as complete.

These controls can be tailored to meet the needs of a particular business or project, and may be modified (by special privilege) during the lifetime of a project.

Other support for Quality Control is provided by the maintenance of history records for all significant PK operations. This provides traceability and gives an audit trail for these operations.

By the appropriate instrumentation of PK operations. it is possible to gather metrics which can be used to improve future planning and estimating.

DATA SHARING

The use of version control ensures that each version of an item is distinct and cannot later be modified; if changes are necessary, a new version must be created.

This regime enables stable versions of items to be shared within a project. PK provides a set of facilities in the form of a double-handshake to control such item sharing.

When an item has been developed to the level necessary for release within the project. the developer may 'publish' that particular version of the item to the desired recipient domains. The users of these domains may then choose whether and when they should formally 'acquire' that item version. That action provides them with read-only access. which might be required, for example. for integration purposes.

The developer of that item version can at some stage decide that it should be 'withdrawn', for example when it becomes obsolete or a bug has been fixed. However. the access to that item version cannot be taken away from its acquirers until they have formally 'disposed' of it. An item version cannot be deleted by a developer until all 'acquirers' have disposed of it.

These facilities ensure dependable access to stable data.

WORK ALLOCATION

In addition to providing read-only access to item versions. it is also desirable to enable and control the allocation of development work. This is achieved in PK by providing a similar set of double-handshake facilities to offer read-write access to item version 'sets'.

A team leader (say) can 'allocate' an item (at a version level) with suitable development privileges to a development domain. The user responsible for that domain can then formally 'receive' that work, and proceed to perform the necessary development work (which might involve onward allocation).

When development has been completed, the developer 'returns' the item to the original allocator who formally 'accepts' it when he is satisfied of its quality. Alternatively the 'return' may be under the control of an independent reviewer.

CONFIGURATIONS

So far, the project control facilities have been described as operating on individual items and their versions. PK also supports configurations of such item versions, which have a collective purpose and a collective status.

These configurations, apart from defining the set of item versions from which they are constituted, are also items in their own right, and can therefore be versioned, shared, and allocated for development. Such operations on a configuration automatically apply the corresponding action to all their constituent item versions. For example, the 'publication' of a configuration, publishes all of the item versions which it 'contains'.

An item version may be used in many different configurations, and any dependencies which are derived through each usage of it are held within the relevant configuration.

Configurations also provide a simplified context in which development tools can operate, in that only one version of any item may exist within the configuration, therefore the tools need not be aware of the versioning system.

PRODUCT STRUCTURE

Clearly PK is a product in its own right and is being sold as such. However, many users have requested specially configured systems, with particular tools or methods supported. One example of this is the HORIZON system being developed for British Aerospace. This PK based development environment will include support for Mascot 3 design and CORAL programming.

CONCLUSION

The Perspective Kernel provides a well-engineered base on which to construct Integrated Project Support Environments. It does this by tackling the underlying problems which made IPSE's necessary:-

- the need to support the whole project life cycle
- the need to help technical management control medium to large project teams of technical staff
- the need for configuration management and version control across the whole project

However, the Perspective Kernel does not dictate the tools or methods the team must use. The project manager can choose the appropriate tools, the appropriate controls, and embed these within his own project-specific, company-specific (or even product-specific) environment.

A major benefit of this approach is that the introduction of an environment into a project can naturally evolve from a simple system (perhaps supporting the programming phase only) into a full IPSE as and when new tools become available.

Equally the Perspective Kernel facilities can be "retro-fitted" to existing tools for projects already well under way, or to help tackle the maintenance problems of existing systems.

DISCUSSION

During the explanation of the Perspective version control mechanisms, the question was asked concerning when an item is in the public domain and is "acquired" by another user. Is there any way to prevent local copies of the item being made? If not, then all dependency links between the items will be lost. Mr. Jackson responded by saying that copies can be made, though obviously this is discouraged. In fact, to overcome the problem of using a non-standard version of an item in a system, it is common for Quality Assurance to insist that a particular preferred version of an item is used in a product.

Professor Randell pointed out that many issues addressed by Perspective appear to be very general engineering problems, not just software specific. He wondered if Mr. Jackson had any thoughts on how this system would cope in non-software areas. Mr. Jackson agreed with this point, and went on to say that Perspective has been used successfully in British Aerospace on a large distributed project involving hardware, software, and firmware where similar problems, particularly concerning versions and configurations, have been encountered.

Professor Habermann drew comparison to the Gandalf system, in which a similar version mechanism is used. However, in that system it was found particularly important to be able to re-use configurations of items, and also to be able to check if particular system parts are already available. Mr. Jackson agreed that this is a powerful feature.

Professor Coulouris was concerned that an important part of the CAD development process involves iteration of the design and implementation. However, this sytem does not appear to have facilities to support this. Mr. Jackson replied that the system has no specific facilities, but iteration is a prime cause of versions of items, for which there is extensive support.

Finally, Professor Atkinson wanted to know how you convince programmers that such developments environments are required, and should be used. In Mr. Jackson's experience, the programmers see the need for them themselves. For example, in systems where strict validation is required, the need for tight control during development is regarded as an aid rather than an unnecessary constraint.