The IPSE 2.5 Project:
Process Modelling as a Basis for a Support Environment

B.C. Warboys,

Department of Computer Science,
University of Manchester,
Manchester M13 9PL,
England.

Published in
Proceedings of the First International Conference on System Development Environments and Factories SDEF1,
May, 1989, Berlin, Pitman 1990.

The IPSE 2.5 Project : Process Modelling as the basis for a Support Environment

Brian Warboys
Department of Computer Science
University of Manchester
Oxford Road Manchester M13 9PL

May 8, 1994

Abstract

This paper examines the work undertaken in the UK Alvey funded project IPSE 2.5[1] from the point of view of the approach taken to the development of the core system. It outlines the reasoning which led to the decision to base the system on a "active process" core rather than the more traditional approaches which have tended to support an entity derived model.It gives a brief overview of the project with an emphasis on the notion of a Process Control Engine. It outlines the basic concepts behind the Process Modelling Language (PML) and the Process Model for Management Support (PMMS) with particular attention being paid to the representation of process change during the execution of a process model. It then illustrates the approach using a case study of an "active process".

1 Introduction

1.1 Introduction to the project

The IPSE 2.5 project is concerned with the problems of how computer systems can be used in the development of software based information systems. The name "IPSE 2.5" is due directly to the identification in the Alvey [alv] programme of three generations of the so called Integrated Project Support Environment - IPSE 1, IPSE 2, IPSE 3. The IPSE 2.5 project lies somewhere between the second generation - characterised by the use of databases to support the core - and the third generation - characterised by the use of expert systems techniques.

However it is not the characteristics of the type of technology which supports the core of the IPSE that is of primary importance to the project. Rather the project perceives that many current IPSEs are merely instances of a store-plus-tools approach wherein attempts, some very extensive, have been made to emphasise support for improved tools interworking by providing a common tools interface. The IPSE 2.5 project is concerned with the deeper integration of the IPSE Users and their attendant processes with sets of *supporting* tools, some integrated others just interworking in a very loose sense. In many cases the Users themselves are also viewed as "tools" of the IPSE.

This paper gives a brief overview of the project as a whole, placing emphasis on one of the main features, that of "process modelling". The problem area of concern to the IPSE 2.5 project is that of computer systems which provide a set of facilities to support people and organisations (as opposed simply to individuals) concerned with all aspects of information systems development. In fact the project is both more limited and more general than this :

- It is more limited in the sense that the scope of an IPSE is ill-defined; it is always possible to think of something else that could be provided in an IPSE. The project has clearly concentrated on a very specific subset of concerns.

- It is more general in the sense that it is clear that not only is there a vast number of different computer systems, but there are also a great many ways of developing them. Moreover, in practice, modern information systems are concerned with the integration of many discrete subsystems and hence their

[1]The IPSE 2.5 project is funded under the UK Government's ALVEY programme and is a collaborative research project. It began in September 1985 and will formally terminate at the end of 1989. It is a collaboration between STC Technology Limited, International Computers Limited, University of Manchester, Dowty Defence and Air Systems Limited, SERC Rutherford Appleton Laboratory, Plessey Research Roke Manor Limited and British Gas plc.

development requires the usage of a number of different methods even within one single systems development. Systems constructed to support the development of such information systems must therefore be, to some extent, general purpose.

The two major aspects addressed by the project may be characterised by the phrases "process modelling" and "formal methods". Process modelling is concerned with the need to provide support for the many processes involved in the production of information systems. Formal methods, in this context, are mathematically based approaches to software development. One of the original aims of the IPSE 2.5 project was to investigate the potential for integration of the support for such methods within a system providing support for the broader set of processes concerned with information systems production. The emphasis in this paper is on "process modelling".

1.2 What is IPSE 2.5

Specifically IPSE 2.5 is a project whose objective is to develop and demonstrate a small number of IPSE's each constructed on the basis of particular characteristics using a generic IPSE tool. The name IPSE 2.5 is generally used to describe such systems. The characteristics which are of principal interest to the project are concerned with advancing beyond current IPSE products in two important areas. These areas have been alluded to above and are specifically :

1. to raise the level of integration within the support system above the store-plus-tools approach by developing IPSEs which have knowledge of the processes by which information systems are developed. The term used to describe this idea is Process Modelling.

2. within this integrated framework, to provide effective support for the use of the emerging development methods based on mathematical formalisms. These methods, known as formal methods, offer very significant advantages to system developers over traditional approaches. However, it is clear that good support is needed if these approaches are to be widely available to industry.

The exploitation of an IPSE 2.5 system will provide a means of holding information about products and projects, it will support configuration management and version control, it will provide means by which existing tools can be used and it will be equipped with state of the art HCI facilities to ensure the achievement of effective man-machine synergy. One of the major challenges of the project is to provide all of this integrated with Process Modelling and the support for Formal Methods in a clear and cohesive manner.

The IPSE 2.5 project is, of course, one which is investigating a number of advanced concepts. The IPSEs being produced by the project have an experimental nature, the properties of which will be evaluated as an important part of the project itself.

2 A review of the historical perspective of the project

In order to understand the reasoning which has led to our current approach to Support Environments it is necessary to examine the influences which such toolsets have had on our development practises. In spite of recent hype which suggests that only recently has the Industry begun to invest in any form of integrated approach to development support we, at ICL in the Mainframe Division at least, have been using such a support environment for nearly 2 decades.

The environment in question is known as the CADES [djp, bcw76] system and has been used since 1972 for the support of the development of the ICL Operating System VME [bcw80]. This system accurately reflects the industrial concerns prevalent in the late 1960's. The focus on these concerns was due in no small measure to the two successful NATO Software Engineering conferences of 1968/69 [jnb, pn] which are arguably responsible for the very term Software Engineering.

Examination of our experiences with this system over the last 16 years lead to some interesting observations on both what has changed and unfortunately what has not during this period. The observations fall into two categories :

- The Industrial Scenario and development concerns of that era.

- The Technology developed to address those concerns.

Perhaps more than anything they are timely reminders at this time, when there is an unseemly rush to standardise Support Environments, that our engineering expertise is still far from being mature. They show us how much has changed (and has still to change) in even our understanding of the Software Development Process. This must lead to concern that in the realisation of the need for an "OPEN" standard we should not hastily define a very closed system.

2.1 Our Industrial Scenario in the early '70's.

Prior to the early 70's ICL (in its various early forms, English Electric,ICT etc.) had produced a number of comparatively small operating systems and associated compilers for the relatively small mainframe machines which existed at the time.

These systems were produced as discrete components and as they gradually grew larger the approach was essentially to treat the enlarged components as collections of programs and to contract out the development of the programs to separate groups of programmers. It was found that this approach was inadequate. Indeed that the cost of the systems compared to earlier systems increased exponentially rather than linearly with their increase in size.

Thus in 1971 when ICL set out to produce a major operating system, VME, for its new range of Mainframes (what became the 2900 and the Series 39 ranges) it recognised that it had a major task on its hands since all the indicators were that the development would never finish. In a sense this has proven to be the case since some 200 development staff have been continually developing the system since that time. The result was to establish a development support project to develop a rigorous system development methodology, Structural Modelling, and a Computer Aided Development and Evaluation System, CADES, to support the methodology.

2.2 The perceived problem circa 1970

At that time the problem was perceived as being product related. Early documents [bcw76] state :

> "The large development team would need to be able to identify and preserve the overall structure of the operating system. Experience on earlier systems had shown how difficult it was to protect a large system from structural decay. The team would need to be able to distinguish between features which affected the overall structure of the system and those which were merely cosmetic.
>
> The methodology and computer aided system would have to facilitate all stages of the operating system development process, i.e. high level design, low level design of implementation, construction, system generator and maintenance. They would have to encourage the codes of good practice which prevailed within the computer industry, i.e. structured programming, data/entity driven design, delayed fixing and binding, design for resilence etc."

2.3 The solution

Structural modelling was first described in detail in [djp] in 1973. It followed the philosophy that system development should be primarily a data-driven top-down process.

The emphasis was on the modular structure of the system being designed to manipulate defined data items, rather than data items being invented to support the encoding of abstract functional requirements. Each code entity was termed a holon, a term borrowed from Koestler [ak] to describe utilities in an hierarchic system.

A language called SDL (System Development Language) was devised to allow the expressions of holon-holon and holon-data relationships. SDL is also used to describe how each holon uses its relationship with other holons and with data items in order to carry out its particular functions. The hierarchy of SDL Holons represents a gradual refinement of the total description of the system,each level being a complete description of the system at that level of design, the level being fixed by the accompanying data tree decomposition. At the lowest levels this use of SDL merges completely into the implementation programming language, S3 (a close relative of the expression oriented language Algol68).

The CADES database was set up to record information about the various holon-holon, holon-data relationship and this was then used as the basis for version control and other management support purposes. It had become, of course, a Support Environment.

By far the most important development was that the database was established as the *only* source of all product components. Source code could only be produced by the execution of an Environment Processor (the EP) which processed design information from the database to produce S3 source code. An early example of process enforcement through the use of tools.

3 Properties of its successor IPSE 2.5

3.1 Observations on the CADES Approach

Within the size and scale of this paper it is only feasible to attempt a very superficial analysis of the approach but the more significant highlights are :

(a) The approach is product structure derived. This leads to some good and bad attributes. The best is perhaps that the architecture of the system plays an active role in the subsequent design decomposition.

The theology of the product approach is reflected directly in the tools used for development. Thus both the architecture and the tools system have survived the test of time. Essentially the same system is still used 16 years and some 3000 man years later. It has demonstrated that Support Environments can have longevity! The worst attribute is that the process of development is necessarily made subservient and thus although the product has not suffered structural decay the process of development has not been able to properly adapt to more modern influences. It is particularly noticeable that the "total life cycle" outlined above made no mention of specification. More importantly the granularity of the database is product oriented and thus the granularity of the supported process is of necessity constrained to the level of the product modularity. Management of the process is therefore conducted at the holon level. Reuse, optimisation, version management and all the other Engineering concerns are thus also constrained to operate at this level.

(b) The approach is essentially one of "Design then evaluate". Support for iteration is provided but essentially the system is oriented towards design capture of new and modified functions followed by the use of conventional (Codasyl based) database technology to invert and evaluate that design. The notion of prototyping is missing and the ability to reason about the design is essentially restricted to structural analysis.

Again the primary goal of managing the avoidance of product structural decay was realised but the granularity of design entities essentially restricted the ability to reason except at holon-holon level.

(c) The system is a closed one. Not really in the sense of current Open Systems concerns since one could easily define a "Public Tools Interface" to the system but in the sense that any tool added to the system is constrained by the core style of schema representation. New tools have to be totally integrated and the resultant costs severely restricted the ability to experiment with new toolsets, to rapidly discard and acquire new tools and generally to adopt a flexible tools strategy.

(d) Many mundane issues are of fundamental importance. In the end most effort was expended on Version Control and it is clear that is of paramount importance that such issues are addressed in such a way that they are all embracing and transparent to tools providers.

3.2 Subsequent conclusions and influence on the IPSE 2.5 approach

(a) It is clear from the above that the principal constraint to a flexible support environment is, not surprisingly, the basic architecture of that environment. The decision to base the granularity of the database at a predominantly holon level had an all-embracing impact on both the approach to an open tools policy and to the type of development process which the tools enabled (and/or supported). In fact the decision to base the environment core on the granularity of the entities to be developed was the key factor. Our experience is that any attempt to base the Support Environment on the entities to be developed rather than the process to be used for the development is doomed to the same constraints as the CADES system imposed.

Clearly a major influence on the IPSE 2.5 [ras1] project was the desire to build a Support Environment with a Process Modelling paradigm at its core rather than an operating system style core with an interface fashioned primarily to entity manipulation.

In particular the IPSE project takes the view that it is axiomatic both that software development is an iterative process and that this process should be managed and changed actively rather than as a transparent side-effect of the product or organisational structure. This implies the need to support considerable process change as development proceeds. The approach is illustrated in sections 5,6,and 7 where a possible model for the management process is outlined.

(b) The "Design then Evaluate" strategy of CADES had highlighted the constraints on better approaches to development. In particular the need to reason about our specifications, designs, implementations is fundamental to good software development. This ability to reason implies a level of integration of process and tools which needs to be determined by the nature of the interaction between the reasoner and the tools and not by the product modularity constraints. As workstations increase in power with improved screen and "tactile" support so the notion of a reasoning assistant will become increasingly tractable [cbj1]. However it is also clear that the Environment is still required to support the conventional level of granularity as represented in the CADES system.

The second major influence on the IPSE 2.5 project was therefore the desire to investigate the integration of these coarse and fine grain support systems. There were also implications on the user interface in terms of the management of vast quantities of fine-grain entities and their subsequent projection via coarser-grain entities, for example to allow the browsing of entities of management interest such as plan impacts.

(c) The closed nature of CADES had highlighted the constraints which such an architecture placed on flexible tool acquisition and disposition. Increasingly there is a need for greater flexibility in creating hybrid design processes and this implies flexibility in levels of tool integration. A simplistic "Operating System" style CASE will not, in itself, by concentrating on the store-plus-tools level of integration enable the rapid construction of Integrated Support Systems as distinct from the support for heterogenous tools systems. OPEN systems imply a speed of change as well as community-wide ownership. It is important to ensure that the community wide ownership desire does not inflict upon us a closed system in terms of development paradigms.

Thus another major influence was the desire for the process based core to be the *means* of flexible and very cheap alien tool interworking, essentially a generic "meta" Public Tools Interface. Hence it is considered of vital importance to an Open CASE system that it provides a standard means of construction of Public Tools Interfaces rather than any one Interface per se.

(d) The notion of Component Re-use had been an early motivator in the CADES system. The hope was that by the use of Database support a generic approach to "Libraries" could be developed, indeed that such commonality would be established at many levels. In practice this did not happen. The means of Specification and then subsequent browsing were not available, but again, even if they had been, the fine-grain integration which such "pattern-matching" implies would not have been realised by a totally "entity"-based core. Again some support for the recognition of the development process in order to enable re-use was required. It was also clear that the development process was such that the re-use of process fragments was as great a contributor to productivity as any design component re-use. CADES did not aid the long period of gestation concerned with the construction of a process,in many cases a heterogeneous assemblage of sub-processes, to solve large scale problems. This need to employ a variety of solution strategies in any given system development is, we believe, of great and growing importance.

3.3 The IPSE 2.5 Approach

The purpose of this kind of Support Environment is to provide the means by which the process of developing, maintaining, supporting and enhancing information systems is made more efficient, in both quality and productivity terms. Traditionally such Environments, be they for the support of programmers or the support of projects, have been considered as tools to support people who have tasks to carry out. The view taken in the IPSE 2.5 project is to stand back from this position of "users and tools" and consider the problem as a whole.

The process of developing, installing and changing information systems is one which involves the co-operative efforts of many people. People are involved in this process because of the intellectual nature of the various tasks. We should not forget that no tool, as yet, can remove the essential involvement of the human being. As Dijkstra [ewd] said in 1972,

> "We shall do a much better programming job ..... provided we respect the intrinsic limitations of the human mind."

The IPSE is thus a means of supporting the whole process rather than being a collection of tools which assist particular activities or classes of activities within that process.

The essence of the integration component of IPSE 2.5 is based on this notion that an IPSE is about supporting the process of systems development, a process in which people (the "users" of the IPSE) play a very significant part. In many ways this is the logical successor to systems such as CADES where the environment is seen as providing the components out of which the process is formed, but in a completely general way which is ignorant of process.

3.3.1 The Process Control Engine (PCE)

At the heart of an IPSE 2.5 system is something called a Process Control Engine (or PCE). The idea is that the PCE is a computer system which provides the (changing) working environments for its users, the people involved in the development process. The PCE is cognisant of the process itself and is thus able to provide the appropriate working environments at the appropriate times. The means by which this is done (and how much can be done) is the essence of the IPSE 2.5 project from a technical point of view, taking particular account of the requirements to support the use of formal methods and in particular, formal reasoning.

3.3.2 Process Description and Composition

It would be possible (but not useful) to build a PCE which only supported a single type of development process. More usefully, it is necessary to provide some means by which a general purpose PCE can be "programmed" to support different processes. In reality, such processes are an amalgam of many separate processes or maybe just constraints on how particular tasks may be carried out. ("I don't care how you do it, but do it by Tuesday!").

The concept of a Process Modelling Language (PML) is introduced to be the means by which such different process fragments can be described and composed. Such fragments also provide the means of "external" tool interworking. The working environments to be provided by the PCE for its users must include all the things that a user "says" he requires for the job in hand. The PML must therefore provide for the descriptions of objects upon which the user might carry out some operations, tools to help him carry out those operations, and means of communicating changes to the working environments of himself or of others. These are minimum requirements of the PML but serve here to illustrate the concept. In "traditional" Support Environments, the database schema (or rather the language in which it is expressed) can be seen as an embryonic PML, often augmented by the command language of the host operating system.

It is important to understand the relationship between a user (i.e. a person involved within an overall development process), PML and PCE. The type of interaction that the user has with the system is determined by the PML description. The PCE is an engine which supports these descriptions. It is of note that the approach being taken is essentially that of explicit process descriptions. It is possible to imagine an approach by which a model is "learned" implicitly by the PCE from actions carried out by users, such models then being available subsequently. The project takes the view that the interactive nature of the people/tools society which the IPSE encapsulates calls for a more explicit approach.

It is however worth noting that explicit models are the basis of "what if" type analyses (or more general property analysis) as well as being used as described here. This idea is one being pursued by the project in particular as part of its work in providing support for project management.

4 The Process Modelling Language (PML)

Two kinds of integration are required in an IPSE, in addition to the usual integration of computing facilitites

- the integration of people's activities
- the integration of people's activities with the computing facilities

The process model deals with both of these concerns; it integrates the ways in which a group of people use computing facilities in support of their software development activities. This development work consists of a number of distinct, concurrent activities, corresponding to the many contributing "Roles". These Roles, and the interactions between them, are thus objects that must be modelled. The resulting process model drives the operation of the IPSE; it determines the actions carried out by the computer system, and offers the user a choice of possible activities, as required by the Roles modelled in the system. By performing an activity, a user contributes to the execution of a Role.

4.1 On Roles and Interactions

This notion of *Roles* and *Interactions* is the basis of process models. The essence of such process models is that they consist primarily of a set of Roles which interact in a meaningful manner. Such a model of a world of Roles is intended to exploit the notion of concurrently executing agents, all co-ordinating to achieve a common goal.

Within each Role are a number of activities that can be carried out by the IPSE (inclusive of its users). An Interaction between Roles on the IPSE is represented by an Action in one Role connected to an Action in another Role.

Thus Roles :

- define the behaviors appropriate to a specific part of the software development process
- exist "on-line" and "off-line" (off-line Roles are IPSE users - note also that Software Tools which are not part of the IPSE are considered to be off-line).

- progress from one state to another until all Actions and Interactions have been completed (note, of course, that Actions and Interactions may be performed more than once in the life of a Role)

Interactions :

- occur between on-line Roles (without user interaction, usually involving the transfer of objects)
- occur between on-line and off-line Roles, ie between an IPSE user and an on-line Role

4.2 Entities and Assertions

In order to allow for the encapsulation of the resources owned by the Roles another type of object is supported : an *Entity*. Together with Interactions and Actions, Entities contribute to the state of a Role at a particular point in time; the values of the Role's resources are an important aspect of that state. Interactions between online Roles generally result in the Entities owned by the Roles (the *resources* of the Roles) "changing hands". The state of a Role thus changes as a result of behavior, in which resources are manipulated and/or transferred to other Roles.

The scheduling of the executions of Actions and Interactions is determined by "start" and "end" conditions. These conditions are predicates which are expressed in terms of the state of the Role, ie the values of the resources and the occurence of Actions and Interactions.

A predicate can be expressed as a Boolean expression but in addition support is provided for *Assertions*, which define a set of such conditions that must hold true for a particular group of arguments.

Thus we have outlined five principal notions of the Modelling language (PML) : Role, Interaction, Action, Entity and Assertion.

Further PML adopts a number of concepts employed in object-oriented languages (cf Requirements Modelling Language [sjg] and Smalltalk [gol]) for the purposes of expressing models. These are the concepts :

- classification/instantiation
- specialisation
- aggregation/decomposition

A class is a group of objects that share a set of common *definitional properties*. For example, the Actions of the class Role, are properties corresponding to all the executable operations which may be performed by the Role.

The Role and Entity notions expressed above are the *principal classes* in PML. These classes are "primitive" object groups, and their behavior is built into the language. They define a set of *property categories* which determine the kinds of property which may be owned by classes of this type. For example, the property categories of the principal class Role include :

- *resources* - the data objects belonging to the Role.
- *assocs* - references to the Interaction objects with which the Role communicates, and any new Roles created by the Role.
- *actions* - Actions and Interactions which operate on the Role.
- categories containing properties which determine the start and stop conditions of the Role.

A process model is a network of objects of the principal classes. Any single class may be a *subclass* of another class (or classes). A process model is built up by creating subclasses of the principal classes, subclasses of those classes and so on in the usual Object-Oriented way. This is, of course, the process of *specialisation*, whereby definitional properties of the superclass are *inherited* by the subclass.

As is to be expected a Role class does not itself exhibit behavior. A Role class prescribes the behavior of an *object* of that class, ie an *instance* of that class. Creating a class instance involves supplying values for the resources defined in the class definition. The Role instance then exhibits behavior - the resources are manipulated by the Actions and Interactions defined for that Role.

5 The Process Model for Management Support

Given the primitive framework for process models described above this section now demonstrates how this framework can be used to construct a richer set of concepts in the area of Management Support. Similar enrichment clearly will also take place to address other IPSE concerns.

The Process Model for Management Support (PMMS) is a set of PML classes, which together describe possible networks of Roles and Interactions. This structure is designed to enable the use oof a raw PML language engine as a working IPSE, in which software projects can be given a technical methodology, can be modelled, and the model subsequently installed. The manner in which this is done is the purpose of this case study; the philosophy of the model is described followed by a demonstration of its use via a scenario based on a simplified version of a possible organisational structure for the development of a set of software packages.

The PMMS is not intended for use in the direct modelling of an organisation.It is more concerned with supporting the enactment of particular processes (eg in developing software). Hence, an entire system containing many instantiations of the PMMS structure, which might represent an organisation, is created from the application of one principle alone: that a new PMMS instance is created to satisfy a goal. There are three fundamental concepts underpinning the PMMS :

- a *Goal* - the description of a requirement (for example a new object to be produced, a change of state to some existing object, or a desire)
- a *Process* - a series of operations which satisfies the Goal, within some constraints identified by the *Terms of Reference* which are fulfilled by completing the process.
- a *Model* - that which represents the definition of the process to be carried out to satisfy a Goal.

A PMMS instance thus takes a Goal as input, creates a Model, and establishes a collection of Roles to execute the Model and satisfy the Goal.

5.1 Model Structure

The PMMS is based on ideas from Organisation Theory, which divide the function of Management into four distinct areas : Peopling, Technology, Logistics and Doing. The PML method requires that we model organisations principally using the notion of Roles. This section illustrates the development of a possible set of Roles which are then specialised to reflect a hierarchical organisation.

Initially there is established a Role class *Managing* which deals with the acquisition and assignment of people, and thus partially models the Peopling function. The Role classes *Technology* and *Logistics* are established to correspond to the Technology and Logistics functions respectively. The Role class *Administrating* represents the interface between Doing and the others.

A PMMS instance can now be created by instantiating the *Managing* Role class, with a Terms of Reference, (*tor*) as input.The Managing instance co-ordinates a network of Roles, which together define and execute a Model for the Goal. Most of its activities are therefore instantiating other Roles, or taking part in Interactions. In order to satisfy the Goal, a Managing Role creates three other Roles :

1. A *Technology* Role, which controls the purely technical aspects of the process (the method used, Quality Assurance, and so on).
2. A *Logistics* Role, which deals with process Modelling (scheduling, resourcing, and so on).
3. A *Administrating* Role, which starts the process by instantiating the other modelled roles, then interacts with them to compile exception reports on the current state of the process. It also implements planned changes to the model.

Each of these four Role classes is specialised for every process in the Management hierarchy. This involves adding behavior specific to the method used, and behavior appropriate to the individual roles. Other process roles can be of two types :

- Instances of a Role class defined in the method (eg *Analyst* for a technical method, and *Financing* for a Business method).
- Instances of *Managing* - these correspond to delegated activities. Hence the *Managing* is the interface to the *super-process*, if one exists.

A *Managing* and its *Technology* and *Logistics* are conceptually at an equal level of authority but represent a seperation of concerns. Thus they carry the responsibility for satisfying the goal of the process co-operatively, but it is the *Managing* alone which can abort the model. Clearly it is necessary that termination can occur at any time in the life of the process.

This relationship between Management Areas and Role Classes is represented diagrammatically in Fig 1.

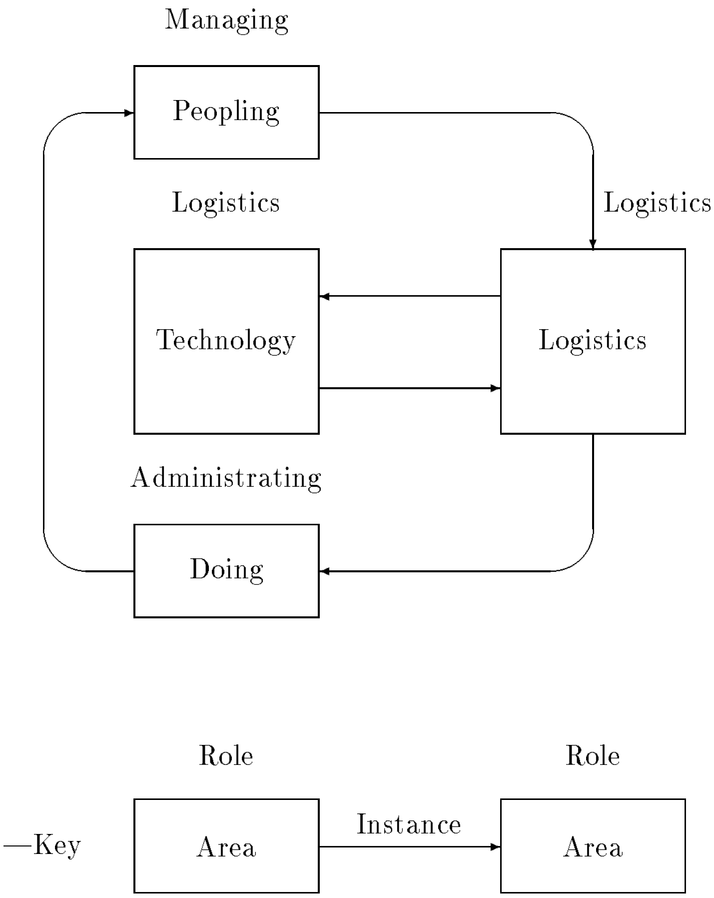

fig1 Role Classes

5.2 Management Cycle

As outlined earlier the management of a process to achieve a goal should be viewed as a cycle, which involves continual refinement of the model throughout the life of the process.It is this active aspect of a life cycle definition which is one important distinguishing characteristic of the IPSE 2.5 approach. The generic cycle utilised by the PMMS paradigm has a number of stages which are listed below together with the PMMS roles which perform each stage indicated in parenthesis :

1. Define the goal (specified in the *tor* of the PMMS and given to *Managing*).
2. Choose the method to satisfy the goal (*Technology*).
3. Model how to resource and schedule roles to meet the goal (*Logistics*).
4. Install the model, ie run process (*Administrating*).
5. Monitor the process (*Administrating*).
6. Evaluate the monitor data (*Managing*).
7. Re-model the process by one of the following actions
   (a) re-designing the method (*Technology*).
   (b) re-assigning resources (*Logistics*).
   (c) or even abandoning the process (*Logistics* and *Managing*).

5.3 Primitive Actions of Change in PML

It is hopefully now evident that the principal means of realising the objective of an “active” process model is through the mechanism of the PMMS Change paradigm. Not surprisingly this leads to the creation of a collection of primitive actions for changing process models *during* execution. These primitives then allow modelling and configuration management to be built into models. An early exampler of the re-use of process fragments. Higher level operations are clearly required to enable the system to be changed in a user-friendly way, but such operations can be easily built from these primitives :

- BaseModelling This Role class has the PML built in classes as definitional properties. Any role instance of this class begins life with a copy of each of the built-in PML classes. The class also has two user action properties *createClass* and *startRole*. A user interacting with a BaseModelling instance may start a new role by interacting through the *startRole* interaction. New classes may be included as the properties of the BaseModelling instance through use of the *createClass*. The *createClass* property class expression names the user action AddClass.

- AddClass This user action invokes a PML editor to create a new object (class). The created class becomes a factual property of the BaseModelling role in which the action occured. The name of the class is supplied through the user action and is the property name appended to the role; the actual class is the property value and is an instance of the class “Class”. The *Naming* is a factual property of the role in which the creation occured. The corresponding definitional property (with the class name as a property name and a class expression of “Class”) is appended to the class of the role in which the creation occured.

- StartRole This user action creates a new role. The created role becomes a factual property of the base model role in which the action occured. The name of the role is supplied through the user action, and is the property name appended to the BaseModelling role; the actual new role is the property value and is an instance of the role class supplied through the user action. The definitional property corresponding to the new factual property (the role and its associated property name) is appended to the class of the role in which the StartRole occured.

- BehaveAs This action requires an object to change its behavior to adhere to that denoted by the behavior class expression. The action requires extra parameters as a number of propertyName, propertyValue pairs; the pairs will assign values to the new properties which the object has acquired as a result of its changed behavior.

6 Case Study - Galactic Software Factory (GSF)

In order to illustrate the utilisation of the PMMS system outlined above there follows a case study of a bold initiative termed the GSF Project. (any resemblance to projects dead or alive is unintentional and is regretted).

6.1 Project Director

This section describes how the Project Director of the GSF Project models the top level of the organisation by

- Defining the Project Method

  The Project method is defined in the following way

  1. The Project Director of the new project initially consists of *pd*, which is an instance of Managing. The goal of the project director is supplied by *base* in the terms of reference *tor*, for example :
     - *Goal* - Convince sponsors to keep supporting the project.
     - *Constraints* - Work to contract. Investment is limited to a certain sum. The project may produce products and market them in addition to working to contract. Investment may change.

     Since *pd* is an instance of Managing (a subclass of BaseModelling) it has its own copies of the classes, which were bound to its resources on instantiation.
  2. The first action is for *pd* to instantiate a Technology, Logistics and Administrating and bind them via the appropriate interactions. These new role instances are called *pd.tech, pd.log, pd.mon* respectively.
  3. This group of roles must now co-ordinate to create a method for the project and produce a model for that method; this is done by *pd.log* asking *pd.tech* to create a suitable method.
  4. The person who is bound to *pd.tech* creates the method by repeatedly calling AddClass, to create the classes which will constitute the new method.
  5. When the method is complete, *pd.tech* tells *pd.log* to BehaveAs the specialisation of Logistics which is appropriate to the new method.

  The Method for the project identifies four activities

  - Recruit staff
  - Brainwash staff
  - Bid for Work Packages
  - Satisfy contracts awarded

  The first of the activities is modelled explicitly. The last three are delegated to other instances of PMMS, known as Business Centres. Hence, in order to execute the top-level model, one instance of *Recruiting* (a PML class defined within the chosen Project Method) and at least three more instances of Managing will have to be created. Suppose that the method is as follows :

  - all levels of the project report to the next highest level at different periods;
    - Business Centre to project Project Director reports are Monthly
    - Work Package manager to Business Centre Manager reports are Weekly

  Note that the Project Director does not constrain the type of reports which the business centre manager may require from the Work Package managers, apart from the fact that they must be weekly.

- Creating the Project Model

  Before it can create a model of the project, *pd.log* must change class from Logistics to PdLogistics.

This is done by *pd.tech* calling BehaveAs, specifying *pd.log* as the object which must change and PdLogistics as the target class. It also binds any new properties of *pd.log* during this operation - these properties include the classes which constitute the PdMethod. The model also binds instances of IpseUsing, ie people, to the process roles, as *assoc* properties of those roles, and tells *pd* to BehaveAs an instance of PdManaging.

- Installing the Project Model

  The model is installed by *pd.log* telling *pd.mon* to BehaveAs an instance of PdAdministrating. The model begins when *pd.mon* changes class. Its effect is to start the various Business Centres and the recruitment process, and to re-configure the PMMS instance as defined by the method.

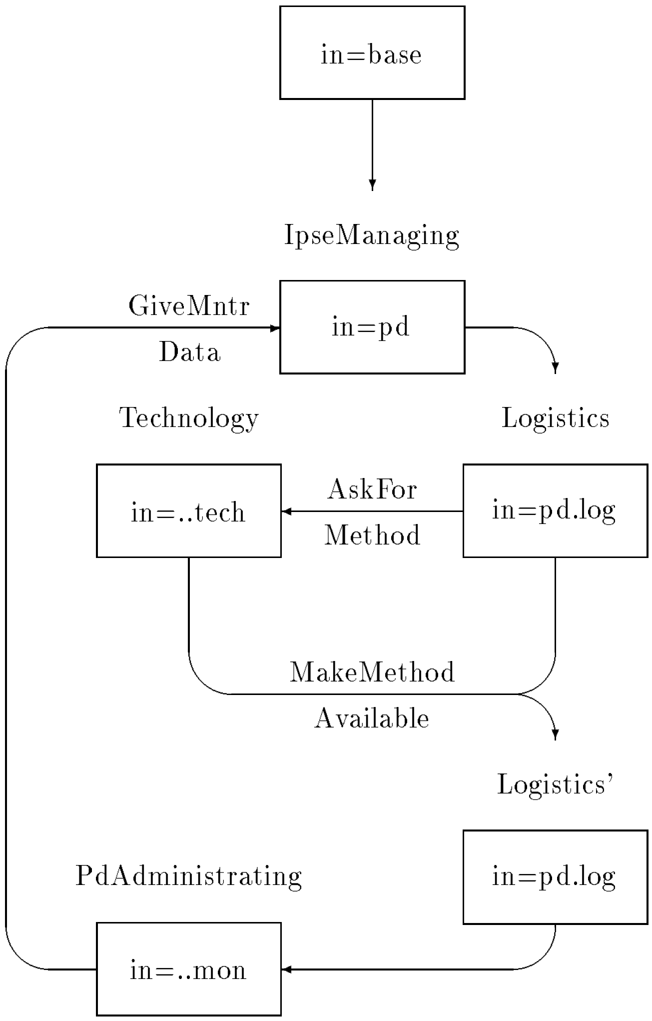

fig2 GSF Mgt Cycle

6.2 Business Centre Manager

This section follows the Business Centre manager responsible for satisfying contracts, from instantiation with no Work Packages identified, to the point where a Work Package is started.

- Choosing the Business Centre Method

  The *pd*'s Administrating role *pd.mon* starts *bc*, an instance of BcManaging, to be responsible for satisfying contracts awarded.The *tor* for the *bc* are given in an analagous way as for the *pd* and appropriate *bc.tech, bc.log, bc.mon* instantiated.

  Working with *tor* alone, since no other guidelines exist, *bc.tech* creates the BcMethod. Summarised by the classes BcLogistics, BcAdministrating, WpManaging - a specialisation of Managing from which all GSF Work Package managers will be instantiated. This class will include actions for the production and transmission of weekly reports to the BcAdministrating. All Work Packages within the business centre are controlled by instances of the PMMS; none of the Work Packages are managed directly by *bc*.

- Creating the Business Centre Model

  The logistics component of the manager is changed by *bc.tech* to become an instance of BcLogistics. Since the initial model produced by *bc .log* does not identify any work, BcAdministrating and BcManaging are not specialised at all.

- Installing the Business Centre Model

  The model is installed by *bc.log* telling *bc.mon* to BehaveAs an instance of BcLogistics. This role has no special actions, since the model is to do nothing until a Work Package is identified. The monthly progress reports required from *bc* to *pd* by the GSF method would reflect "no activity".

- Starting a Work Package Within the Business Centre

  As soon as some work for the Business Centre is identified by *pd*, say a Work Package to write a Theorem Proving Assistant (TPA), the *tor* of the Business Centre is altered by *pd* in the, by now familiar, style

  - *Goal* - Satisfy the contract for a TPA by performing a Software Development Project
  - *Constraints* - Supervise the TPA Workpackage within the constraints defined for *bc* (weekly reports etc)

Having received the new *tor*, the *bc* creates and logs an exception report which is then sent to *bc.log*. This is an instruction to *bc.log* to re-model the activities of the Business Centre. In this case, the changed model will instantiate a new Work Package to implement TPA. The action of re-modelling is similar to the modelling already described :

1. *bc.log* specialises the class of *bc.mon*. This class will support the new TPA Work Package.
2. The new model is invoked by *bc.log* telling *bc.mon* to BehaveAs this new class.
3. After *bc.mon* has changed class, it executes its new properties and thus instantiates a Work Package to handle the TPA. The WpManager for this Work Package is called *wp*.

6.3 Work Package Manager

This section describes how a Work Package manager models a Work Package. We might hope that the approach would be something like the use of VDM [cbj86] as a Specification Method and Smalltalk [gol] as the Implementation Language.

- Choosing the Work Package Method

  The *bc.mon* instantiates *wp* with a *tor* which tells it to run a project to build a TPA :

  – *Goal* - Build a Theorem Proving Assistant (TPA)

  – *Constraints* - Implement the system in Smalltalk. Use VDM as the specification method. Other methods may also be suitable.

  The Work Package is started by *wp* instantiating *wp.tech, wp.log, wp.mon*. The method to be used is supplied by *wp.tech*. Since the *tor* recommends VDM, *wp.tech* supplies that method.

- Creating the Work Package Model

  The model is produced as follows :

  1. *wp.tech* tells *wp.log* to BehaveAs an instance of VDMLogistics
  2. *wp.log* produces the specialisation of VDMAdministrating which is the Work Package model. The model also includes *assocs* to IpseUsing roles (effectively Staff assignment). The Work Package model also "knows" about monitoring points in the life of the Work Package. eg When we have completed a refinement step we must identify the relevant proof obligations.

- Installing the Work Package Model

  As in the previous sections, the model is installed by starting an instance of the specialised Administrating class. In this case the Work Package is initiated by *wp.log* instantiating *wp.mon*, an instance of TPAAdministrating.

- Re-modelling the Work Package

  This section discusses how the TPA Work Package might be re-modelled, given that further Specification Refinement steps are required. It is in the nature of VDM that further refinement steps are required during the Specification process so this event is realistic. The change is achieved by *wp* telling *wp.log* to re-model the Work Package, with two refinement steps to be delivered. The changes to the model are installed by *wp.log* telling *wp.mon* to BehaveAs TPAAdministrating2. The new model is then automatically invoked, without affecting the running Work Package roles.

There follows a diagram of the Work Packet Management cycle specialised for the TPA Work Package and some sample PML which illustrates the construction of a Logistics instance and a Role instance. They should all, by now, be self explanatory.

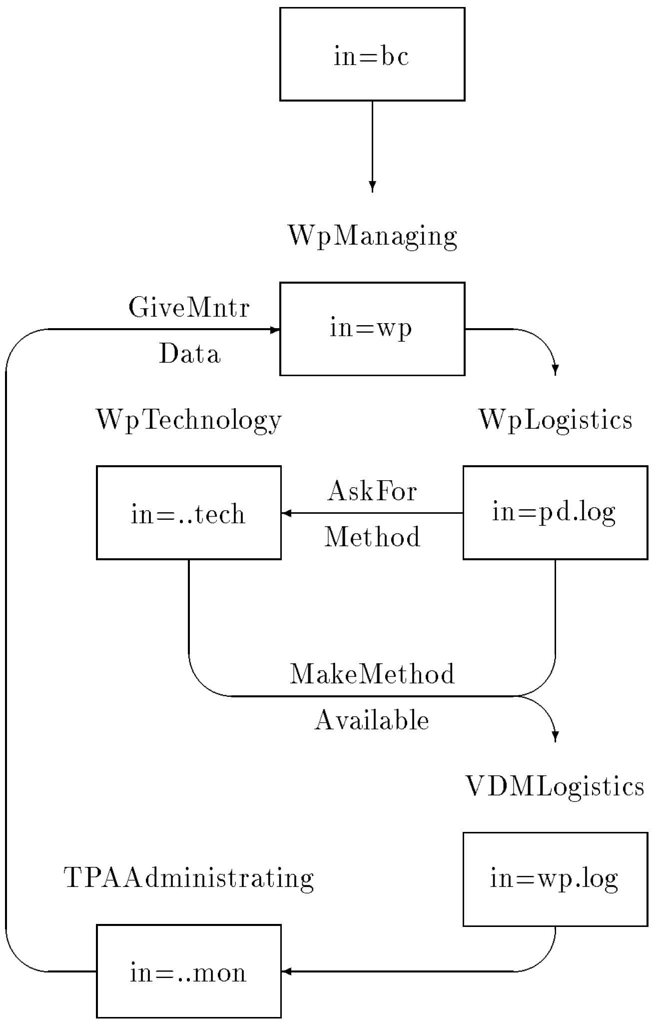

fig3 TPA WP Mgt Cycle

Some sample PML to illustrate the model :

```
---------------------------------------------
VDMLogistics isa Logistics with

    resources

        Designer : Role class
        VDMAdministrating : Role class
        VDMLogistics : Role class
        VDMManaging : Role class

        Interview : Interaction class
        AuditorApproval : Interaction class
        CommitteeApproval : Interaction class

        WriteSpec : Action class

        SpecPackage : Entity class
        VDMSpecification : Entity class
---------------------------------------------
Designer isa Role with

    resources

        spec : VDMSpecification
        readyForApproval : Bool

    preconds

        torDefined : spec.tor ne nil

    actions

      getHelp : Interview
        through designerPort (package = spec)
        when (noHelp : spec.expertise = nil)

        workaBit : WriteSpec (doc = spec
                readyFlag = readyForApproval)
        when (tryAgain : spec ne nil .......

      logSpec : OneToOne through give
                        (docIn = spec)
        when (readyToLog
              : spec.wpApproval = true)

    termconds

        specLogged : logSpec ne nil
---------------------------------------------
```

```
Interview isa UserInteraction with

     ports

          designerPort : communicate {package}
          expertPort : IpseUsing
          moderate : communicate{issues}

     grams

          package : SpecPackage
          issues : Document

     parts

          viewSpec : ViewObject (
               object = package
               class = SpecPackage)....

     stopconds

          finished : raiseIssues ne nil and
               noIssues = true
---------------------------------------------
WriteSpec isa UserAction with

     in

          doc : SpecPackage
          readyFlag : Bool

     parts

      writeVDM : magic alien VDM editor tool
      signalApproval : Assign (
          from = true to = readyFlag )
          when (finished : writeVDM ne nil )

     stopconds

          specWritten : readyFlag = true
---------------------------------------------
```

6.4 Extensions to the case study

Within the scope and scale of this paper it was only possible to attempt a very superficial case study. It has concentrated on demonstrating the means by which Specialisations of Role Classes and the subsequent creation of Role Instances can be used to model a heirarchical organisation. It should also be clear that the detailed development of, say, the model of the VDM process can proceed in parallel with and largely independent of other aspects of the project's processes.

This is but one instance of the use of Specialisations within Process Models. Other topics which are of primary importance such as Monitoring and Configuration Management are addressed by similar means. Thus the generic approach obviates the need for specific discrete implementations of such strategies. This is clearly important since there is a requirement to modify these strategies and to allow for differing sub-strategies with respect to such concerns in much the same way as with other aspects of Process Models.

6.4.1 Monitoring

Monitoring consists of measuring what has been achieved by a project, and the cost of that achievement. Such measurement could include :

- *Functional monitoring* : which notes which process deliverables or activities are complete.
- *Temporal monitoring* : which notes the amount of time spent by the process on different activities.
- *Financial monitoring* : which notes the amount of money spent by the process on different activities.

This list is not intended to be exhaustive, but gives a flavour of the different styles of monitoring which the PMMS must support. The secret is that by allowing specialisation, the PMMS allows any kind of monitoring, but implicitly.

An approach can be easily demonstrated within the TPA Work Package. The WpAdministrating will monitor the completion of each specification. Functional monitoring is achieved by specialising VDMAdministrating to note when a specification is approved. Since specifications approval is already a part of the function of this class, no further interactions need to be added to the model.

The effect of these specialisations, for example to add a Milestone entity, is to record the date on which each specification is approved by the Work Package manager in the TPAAdministrating role. The forecast dates for each Milestone would have been set during the construction of the model. Reports are then built by the Work Package's Managing role, by periodic inspection of the state of the Milestone's entities.

To invoke this monitoring function *wp.mon* is told to BehaveAs TPAMonitoringAdmin when the model is installed. This extra monitoring behavior is then included in the Work Package model. Similarly Temporal and Financial Monitoring can be added.

6.4.2 Configuration Management

Configuration Management or "Management of Change" is a primary concern of the IPSE 2.5 Project. It views the problem at three different levels :

1. Change in a Process Model. Supported by the concepts of :
   - Changing an object by changing the value of a property or by the addition or deletion of a property.
   - Specialisation of a Class by the addition of new definitional properties (property name, class expression pairs) or by the specialisation of existing properties (subclassifications of the class of a property).
2. Change to a Process. Achieved by changing the activities of a process model and supported by the concepts of :
   - An explicit remodel during its initiation cycle. The Managing role instance instantiates a Logistics and Technology and causes the technology instance to re-model the PMMS by making the logistics instance BehaveAs a specialisation of Logistics.
   - Further changes to the project models as defined by the Managing. The re-models occur as a result of evaluations of monitoring data provided to the Managing role by Administrating.
   - The final type of re-model occurs when Managing is changed (by a super-PMMS) to BehaveAs a new class - thus to exhibit behavior, according to the revised model.
3. Change to a process Product. Achieved by the creation of a new product variant.

   Here the concerns are with the concepts of version control, edition numbers, variants etc. A process model provides a strict context for all the activities of a process; for example entities are always explicitly named. As a result conventional concerns about version control are to a large extent irrelevant. Essentially the approach taken is that specific local names are used in the isolated contexts, but for coordinated activity (such as an interaction between client and vendor over an error report) the context differences must be resolved. Interactions between the client and vendor must be objective, ie the *process* model identifies the faulty component through parts-list databases, client serial numbers etc. Hence it is the *process* of Product Change management rather than the discrete naming of variants which is the means of Version Control. Since all change is made explicit and the context for each change is also explicit there is no need for an underlying implicit revision numbering scheme.

6.5 Summary

This Section has outlined an example of how an organisation might be modelled using the PMMS together with the primitive actions required to support model change. Further it has explained how configuration management and a range of different styles of monitoring might be supported. The structure of the modelled organisation is shown in fig 4 below.

It has been a simple illustration of the way in which the use of specialisation within the PML framework supports the vital process of *change* in an active process support environment.

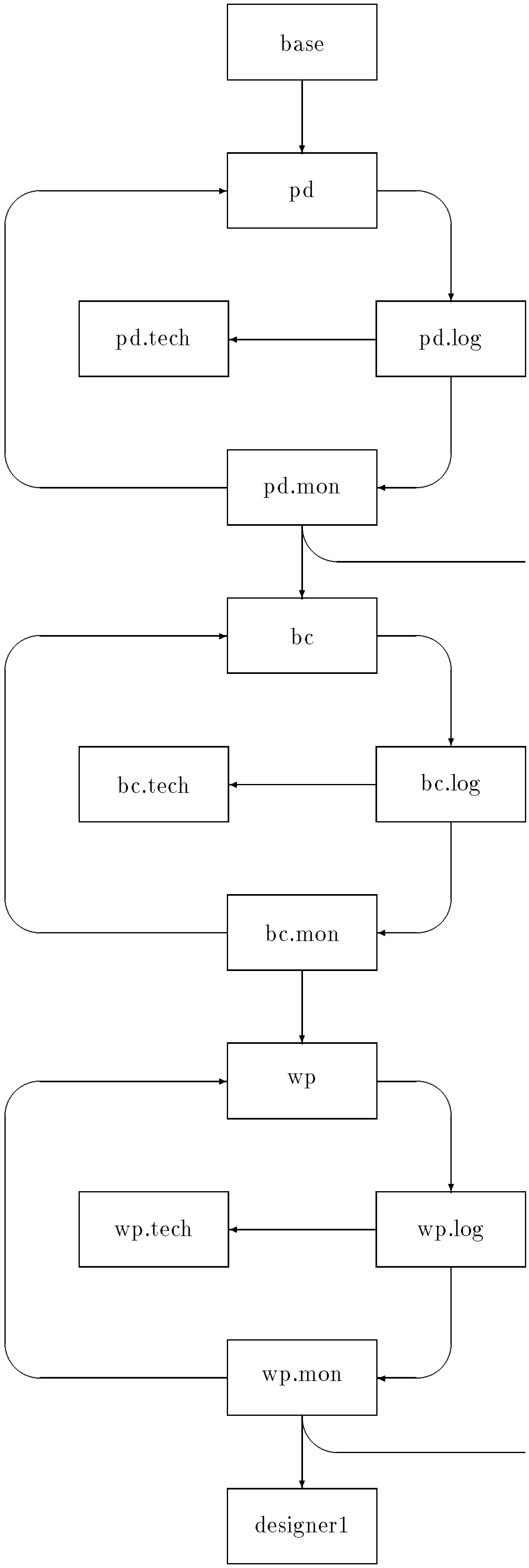

fig4 GSF Scenario Summary

7 The relevance of the IPSE 2.5 approach to Software Engineering processes.

To understand the relevance of such a generic approach rather than the stylised "operating system" core beloved of many CASE offerings there is a need to examine the current context within which we need to engineer software.

Major changes have taken place in the Industry Structure, the Technology and the Market since we developed the CADES system. Most of these emphasise the need for an Open system, in a Process sense, and in terms of flexibility of Tool interworking.

The Industry is, in general, no longer concerned with the development of discrete software components. Instead the major concern is with the "glueing" together of components to provide the IT component of some larger ecosystem. There is a need to recognise the endemic nature of IT, to recognise that our traditional software component is but one component of a much broader control system. Further there must be a recognition of the need for the support of mixed componentry, much of it never to be specified in terms to which our software development methods can sensibly relate.

There is a need to recognise the ever increasing variety of methods, languages and toolsets, to recognise the need for support for system development "in the large" and for component re-use of a wide variety of component types. In particular this paper has attempted to highlight the need for re-use of process model fragments as being of paramount importance.

8 Acknowledgements

This paper has been to a large extent a flagrant plagiarisation of many IPSE project papers. My thanks in particular go to Clive Roberts, Alun Jones and Keith Harrison-Broninski of Praxis [pra] who provided the mini-Praxis model from which the GSF example is constructed. To Bob Snowdon of STL for both base material [ras2] and a most helpful set of comments on the original draft and finally to all the members of the IPSE 2.5 project who unwittingly provided me with material.

References

[alv] Alvey Programme Software Engineering Strategy, November 1983

[djp] D.Pearson "CADES" Computer Weekly,July 26th,August 2nd,August 9th 1973

[bcw76] B.C.Warboys and G.D.Pratten "CADES - Principles" Seminar Oxford University February 4th 1976

[bcw80] B.C.Warboys "VME/B a model for the realisation of a total system concept" ICL Technical Journal November 1980

[jnb] J.N.Buxton and B.Randell "Software Engineering Techniques" Report on NATO Science Conference October 1969

[pn] P.Naur and B.Randell "Software Engineering" Report on NATO Science Conference 1968

[ak] A.Koestler "The Act of Creation" Macmillan 1964

[ras1] R.A.Snowdon "IPSE 2.5 Technical Strategy" IPSE 2.5 project document 060-00131-2.2

[cbj1] C.B.Jones and R.Moore "An experimental user interface for a Theorem Proving Assistant" IPSE 2.5 document SE13/29/234

[ewd] E.W.Dijkstra "The Humble Programmer" CACM No 10,Vol 15 1972

[sjg] S.J.Greenspan "Requirements Modelling; A Knowledge Representation Approach to Software Requirements Definition" Technical Report CSRG-155 University of Toronto, March 1984

[gol] A.Goldberg "Smalltalk-80 The Interactive Programming Environment"

[pra] K.Harrison-Broninski, A.Jones and C.Roberts "A Scenario for Management Support in IPSE 2.5" IPSE 2.5 document 060/00164/1.0

[ras2] R.A.Snowdon "A Brief Overview of the IPSE 2.5 Project" IPSE 2.5 document 060/RAS026/1.1

[cbj86] C.B.Jones "Systematic Software Development Using VDM" Prentice-Hall 1986