An Implementation of the ISPW-6 Process Example

Ian Robertson
email: ir@cs.man.ac.uk

Department of Computer Science
University of Manchester M13 9PL

Abstract. This paper documents some of the work involved in describing the ISPW-6 process example in terms of graphical models and in terms of a modelling and enactment language PML. It further describes experiences in instantiating this model in ICL's Processwise Integrator support system, and outlines the lessons learnt and directions for future work.

1 Introduction

There are numerous technologies, in various states of development, which attempt to support the software process. It is not an easy matter to make comparisons between technologies; not only because of the variety of approaches but also lack of a familiar and comprehensive model on which to base comparisons. In order to remedy this state of affairs, the ISPW-6 [7] example process was formulated. This was developed at the 6th International Software Process Workshop, and hereinafter will be referred to as the ISPW-6 Example, or simply as the example process.

After a very brief introduction to the enactment technology, the work of modelling the process example is described: both in terms of the different models needed and in terms of the process of modelling. At the end of Section 3 there is a description of the instantiation of the model and its enactment. The lessons learnt are discussed in Section 4 with Conclusions addressed in Section 5.

2 The Processwise system

ICL Processwise comprises a suite of services designed to support organisational process activity in a number of ways. The 'Business Modeller's Workbench' offers a tool to model and to simulate process behaviour; the 'Integrator' is a process integration and enactment system; and the 'Guide' describes a particular methodology designed to exploit the features of these technologies. The technology of immediate interest is Processwise Integrator (PW Int) [1].

The system originated in the Alvey IPSE 2.5 project which concluded in 1989. ICL subsequently took up the software rights, hardened the system, and implemented the Process Control Engine in the Unix environment. As a first step to commercial release it was renamed Process Support System. After STC's sale of ICL to Fujitsu, the development team became known as the Process Support Centre and the technology was further developed and became a part of the Business Process Re-engineering strategy known as Processwise. The PCM was implemented in DRS 6000, and with other changes the Process Control Engine (PCE) became the Process Control Manager (PCM), and the language Process Modelling Language (PML) became Process Management Language. The current release of the system is ProcessWise Integrator 2.

Liu and Conradi [3] report that the paradigm of PW Int is closest to that of Arcadia [13]. The semantic model is the process programming language, there being no generic model, and the application-specific model is defined as a program in that language. The system takes the role/activity/interaction notion of defining the mechanism for causing transformation events and provides for their enactment in the same context. It provides a framework within which tools can be effectively co-ordinated to service a process representation of an information systems development methodology.

It is thus a process-centred environment comprising three elements: the PCM , the UI Servers, and the Applications Servers.

The environment holds a model of the process network described in PML [9, 10]. The language identifies four principal classes: entities (records), activities (procedures), roles (encapsulations of data and behaviour) and interactions (corresponding uni-directional communications channels). These principal classes, along with numerous primitive classes, can be used to define the process network. There are three language interfaces which together control the evolution of the model: they allow a class definition to be compiled into a class value, a new instance of a role class to be created, and a role class value to be re-defined.

The heart of the system is the PCM; that component which identifies, locates, and schedules activities which can be executed. It does this by scrutinising each role in turn and examining the guard property of each action which determines *when* that action is allowed to be executed.

Process participants communicate with roles through the medium of a 'pseudo role' known as user agent. The User Interface representation is hierarchic, offering, at the highest level, an agenda of the roles bound to a participant's user agent. Each of these opens into an action agenda allowing selection by the user. This is shown in Figure 5 and described more fully in Section 3.9.

Application integration comprises the ability to pass data as byte string or object to an application running in another environment, to interact with it, and to obtain outputs.

3 Implementing ISPW-6 in PML

3.1 Overview of model architecture

The process example is of a nominally generic process commonly found in organisations concerned with the development and maintenance of software systems. It focusses on activities associated with making planned changes to single specific modules or units which comprise a single software system. No information is provided about participant skill levels, whether the enactment should cater for novice or expert or both, of the software system itself, or of volumes. The process is described clearly and concisely in narrative form as a list of activities defined with inputs, outputs, purpose, and responsibility. This effective statement of requirements corresponds to Feiler and Humphrey's [5] Process Definition level of abstraction, and to the PM2/PM3 meta-process level of Conradi et al [2].

A number of writers confirm the difficulty of representing process activity in a meaningful and concise manner. Curtis [4] confirms the four perspectives widely accepted as being needed to describe a process: organisational, functional, behavioural, and informational, but it is not an easy matter to obtain clean abstractions of these perspectives with the tools currently at our disposal. Conradi et al take a rather more pragmatic approach and recognise at least five sub-models:

1. Activity model. This is the executable model.
2. User model. Describing the service provided to the process participants by the system, and the corresponding obligations placed on them.
3. Organisation model. The model of organisation role structure and functional relationships which associate these roles.
4. Product model. The data structure and relationships.
5. Tool model. Description of tool architecture.

This latter was considered more appropriate as a framework in which to describe the different facets of the subject process and the models needed were not exactly as postulated in [2]: the nature of the process is such that there was no need for a tool model, and, because of the focus on enactment, there was a distinct need for a model of the work infrastructure or environment appropriate for participants; this was called the Environment model.

3.2 Methods and tools

The work described herein does not fall neatly into the phases of the PM1-5 meta-model [2], as the focus of the ISPW-6 example is on the implementation of a model already fairly well-defined in narrative form, with adequate information to begin the PM3 Customisation phase. PADM (Process Analysis and Design Methodology [16]) provides a coherent strategy and techniques for much of the area of PM1 through to PM4, and indeed outside of this framework, identifying the need for a 'baseline' model, stressing that once a 'target' process (or even processes) has or have been identified, the implementation route will of necessity be from 'what is' to 'what ought to be'. The ISPW-6 example assumes a 'green field' implementation, i.e. there is no need to take into account current work practice and cultural influence.

Certain of the representation methods advocated by PADM were also used in this study: the first using the role-interaction paradigm of process [6] to identify user roles (i.e. sociological roles within the organisation, commonly aligned to job descriptions), to map tasks or activities to these roles, and to identify at a coarse grained level discrete states and associated events of the process. The resulting diagrams are known as RADs [8]. The second was the use of the activity diagram (AD) to map activities independently of roles [1]. There is another type of diagram in

[1] A PC modelling tool RADitor is available for role-activity diagrams (developed by Co-Ordination Systems), and the activity diagramming tool is a component of Process Modeller's Workbench being developed by the Informatics Process Group.

PADM known as dependency diagram (DD) which provides a high level abstraction of the essence of complex processes. This was not required in this simple example. The Methodology does as yet provide no guidance for implementation or for selection of support platforms.

3.3 The User Model

In this study the model took two forms. The first was a RAD model of roles, tasks and artifacts shown in Figure 1. It provided a very concise representation of the process, however the greater detail needed to structure an implementation was provided by the second form: it was much more fine-grained, and comprised narrative lists of user activities which might be ascribed to these roles. For example a designer would want to be notified of an impending task, to be made aware that documents were ready and available for use, to be able to work possibly at intermittent intervals, to dispatch the work item and finally to be notified if re-work was necessary or if the item had received approval. We would assume that the task would not be complete until the design had been approved in review, so the facility to iterate would be necessary.

Role activity Diagrams define the scope of role behaviour. When considering organisation roles, they indicate the different activities which can take place and the dependencies which exist between them.

The meaning of the diagram can be clarified by taking the example of the organisation role of Design Engineer.

Once notified of a task, the Design Enginer can work on either Modify Design, or Modify Source Code. Once the code has been modified and passed to Design Review, the activity goes into a 'waiting' state (identified by the tail). If the code is approved, the activity Modify Design is completed and ceases to exist. If Code Feedback arrives, the activity can be re-invoked by the Design Engineer to address the feedback issues. Another kind of dependency is illustrated at completion of Modify Source Code. Here, the Source Code cannot be despatched to QA Engineer until the Approved Design is available.

The only significant departure from the example process is that joint responsibilites have been re-interpreted into unitary responsibilites. The implementation technology has, as yet, no mechanism for interpreting joint responsibility in terms Processwise Integrator technology.

It should be emphasised from the outset that we are dealing with two distinct types of 'role'. There is the organisation role referred to here, which can be conveniently referred to as the user role; and the unique definition of role in ProcessWise (single threaded, defining scope of entities, forming the basis of the user interface, unit of change) which can be called a system role, or in PW Int terminology, an on-line role.

An example of the activity list is provided in the description of the application model for the Modify Code task on page 8.

3.4 The Environment Model

The process example was well defined, but to make the enactment meaningful in the context of an organisation some attention had to be paid to the support or infras-

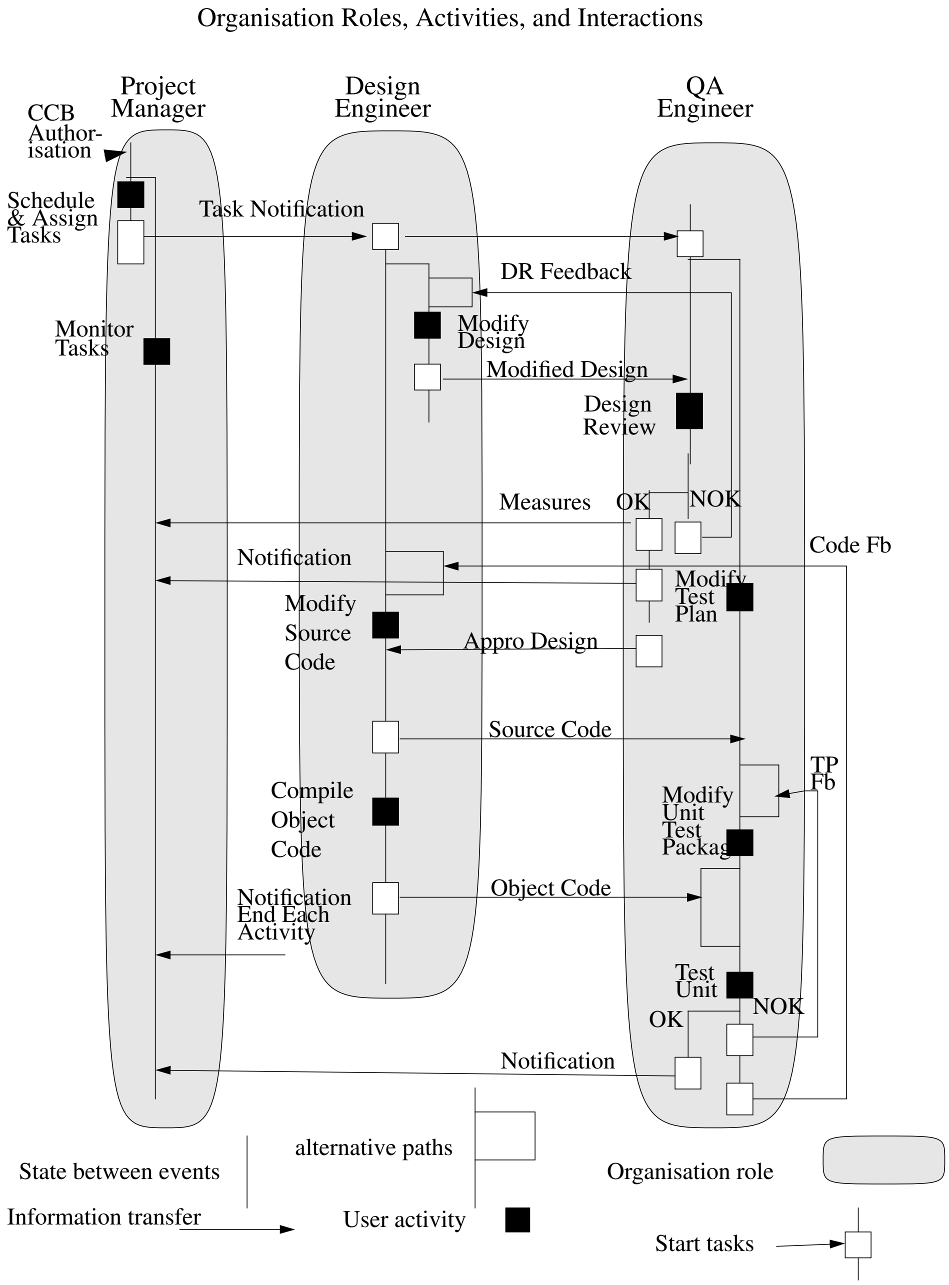

Fig. 1. The User Model

tructure necessary for the process to be enacted. This helped ensure a reasonably realistic implementation for participants, and assisted in throwing light on some of the issues which could arise in a real-world implementation.

It was found helpful to encapsulate certain 'housekeeping' activities into system roles which were permanently bound to participants . These roles supported service activity rather than task activity and were referred to as Admin roles. They are discussed on page 13.

The environmental aspect which was selected for inclusion in this implementation was that of the library service. To be effective a process enactment system would have to interact with the library facilities used to hold these data items, and this is represented in Figure 2.

3.5 The Organisation Model

This was a simple reporting hierarchy with a single manager having responsibility for and control over a group of software engineering staff who could be identified as primarily either design engineers or QA engineers. It clearly indicates that as the process spans only a single group of users, the PM is responsible for controlling both the working of the process and the evolution of the process. The other significant concern in the organisation, the mapping of role sets [12] of group responsibilities such as the Design Review team to the organisation structure, was not addressed. Delegation issues were not present in the process example.

3.6 The Product Model

As product is not the primary concern in this study, this model is trivial. The process example only considers a single system whichyou comment o comprises unidentified modules or units each of which is composed of design, object code, source code, test plans, unit test packages under version control. This control was modelled in the library service.

Notwithstanding this simplistic situation, the issues surrounding this kind of model and its relationship to the associated process model are current research topics.

3.7 The Activity or Application Model

The use of the term 'application' is preferred in the context of the executable model as it clearly refers to a system model, whereas 'activity' might be either real-world activity, or system activity and can also get confused with activity diagrams which are useful for structuring real-world activities.

This is the model which is ultimately enacted, and in its final form is expressed in the process definition language[2] PML for execution in ProcessWise Integrator system. The development of the model, of what is in essence a process management

[2] The term 'process definition language' is possibly more appropriate than 'process modelling language' in that it reflects more accurately the departure from the original ideals of the language.

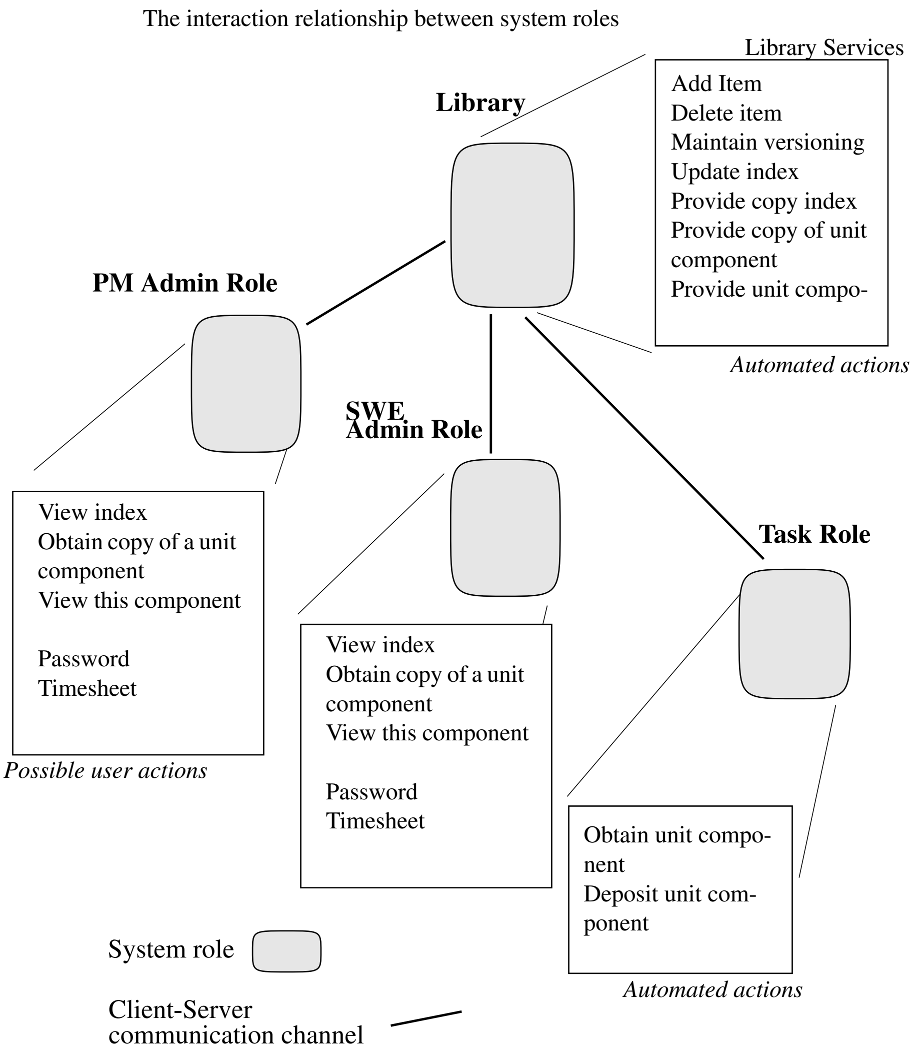

Fig. 2. The Environment model

system, was driven by the nature of PML and the properties of the support system. Of paramount importance in PW Int is the need to identify from the outset the appropriate system roles and granularity appropriate for the application. It is not impossible (but highly unlikely) that system roles could map directly on to user roles, and at the other end of the spectrum a role might represent but a single activity. However the most likely situation is that the behaviour will somewhere between these two extremes. Key considerations in defining system roles are:

- Need for run-time concurrency
- Scope of data entities
- Requirements of user interface
- Desired scope of change

The user model has been transformed to reflect these criteria and this is shown in Figure 3 as Application Model 1. There is a small extension to the Requirements to better illustrate the receiving of a modified design. Review system roles are provided to enable users to participate in the review process prior to a review meeting at which decisions might be made. On the other hand, CCB was not modelled as a system role because it increased complexity without contributing to understanding. The control process for the application model, really a component of the meta-process itself, is shown in Figure 4.

These diagrams formed the basis of describing the process in PML. The system roles were constructed of three basic overlays utilising the inheritance properties of the language. The essential behaviour of *all* task roles is defined in *WPRole*; these were : to maintain and display when required certain task information (identified as *a_scheduleDates* below, information is bound on instantiation and subsequently changed by a Re-scheduling action); to provide a facility to display a message when and if the task was changed by higher authority such as the PM (identified by *displayMessage* changed by a Hold/Restart/Cancel action), and to dispatch a message to PM on conclusion of the task (*sendIt*). The PML for this behaviour is illustrated below by the definition of *WPRole* as a sub-class of the pre-defined PML role class *Role*.

```
WPRole isa Role with
classes
assocs
        doneGP   :giveport DonEnt
resources
        wpEnt   : WPEnt
        thisRole:String
        whoiam  : String := ''
        doneEnt : DonEnt
        kill    : Bool := false
        schedFormat : collof UIField:=collof UIField(
               UIField(name='startDate', label='Schedule Start'),
               UIField(name='duration', label='Duration'),
               UIText(name='requirements', label='Requirements'),
               UIField(name='notes', label='Note'))
```

```
        seeMessage     : Bool := false
        onHold         : Bool := false
        messageString  : String :='New Scheduling Information '
actions
        a_scheduleDates:{
                UserAction(agendaLabel='Task Information');
                ModifyNow
                       (agendaLabel='Task Information',
                       object = wpEnt, readOnly=true,
                       format=schedFormat)
                        }

        displayMessage:{
              ViewResourceNow
              (agendaLabel='  Message', object=messageString,
              label=whoiam);
              seeMessage:=false
                       }
        when seeMessage

        sendIt:{
                Assign(from=whoiam, to=doneEnt.task);
                Give(interaction=doneGP, gram = doneEnt);
                kill:=true
                }
       when nonnil doneEnt

termconds  kill
end with
```

The specialisations appropriate for each particular kind of task were then defined as sub-classes: for example the Modify Code task role (*MSCRole*) is a sub-class of *WPRole* which incorporates the following behaviour:

– Create an object *scEnt* to represent source code.
– Allow modification *scEnt* using a text editor.
– Allow dispatch of a copy of this code to Modify Unit Test Package role only after the approved modified design has been received by Modify Code role from Review Design.
– Simulate compilation of the code in *scEnt* into object code.
– Allow dispatch of this object code to Test Unit task role.
– Provide for handling of feedback from Test Unit task role.
– Provide status messages.

Finally, the definitions of the library service client were instantiated as a sub-class of *MSCRole*, called *MSCLibRole*.

The implementation of the library facility shows how a network of roles and interactions can be constructed to work in a client server manner. The library is the server. It understands a number of library requests which are sent to it using

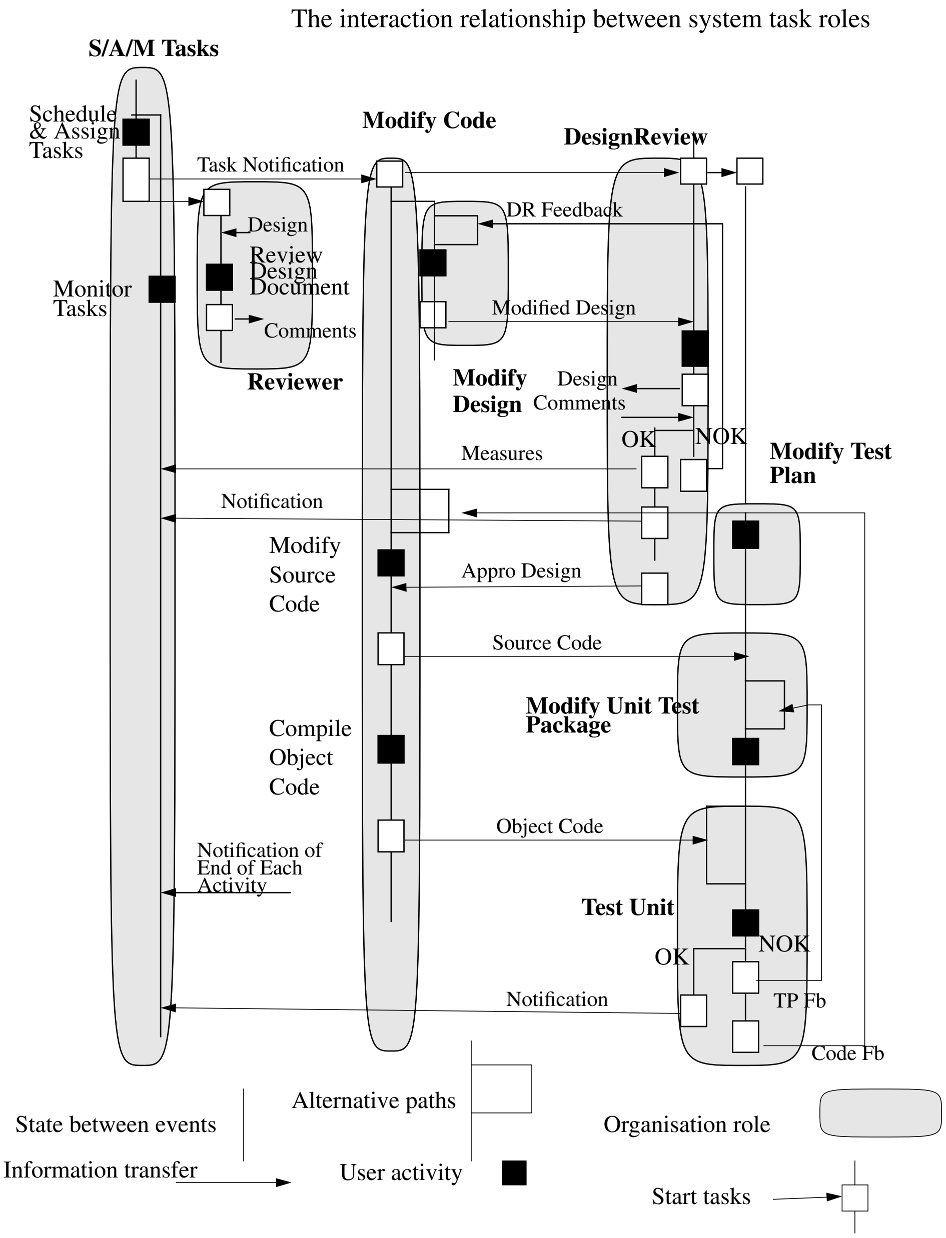

Fig. 3. Application Model 1

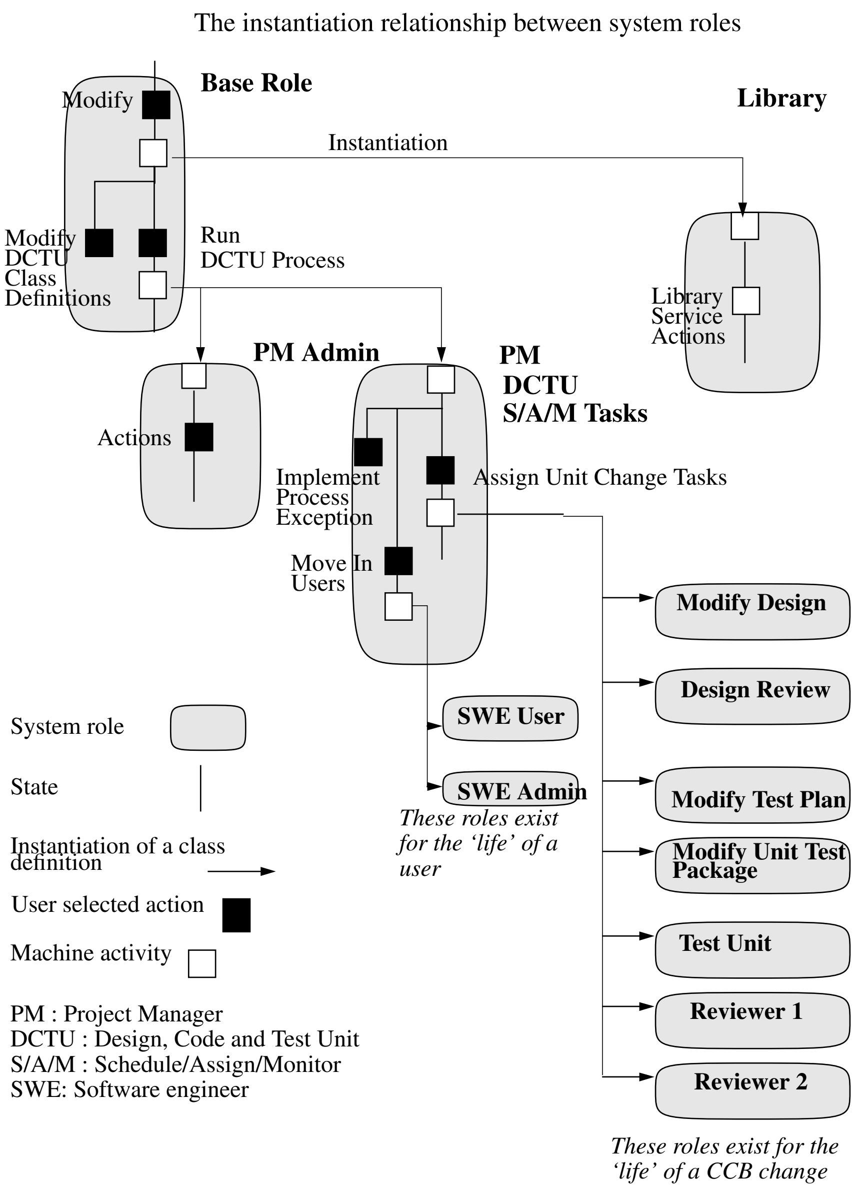

Fig. 4. Application Model 2

an interaction. All roles which use the library can give requests to this interaction, only the library will read them. Each role which uses the library is a client. When it sends a request to the library it includes the giveport of the interaction which the library will use for the reply. The client role will subsequently receive the library's reply on the interaction's takeport.

With this well defined interface, the system roles which use the library do not need to know any details of its implementation. They only have an interaction giveport to which they give library requests. (To be a library using role it will either be given this giveport on starting or at least before its first request.) Indeed they cannot tell if the library is implemented by one role, a collection of roles, or one, or more, roles interfaced to some external tools. The current implementation has the library as a single role which handles requests one at a time.

The data held in the library is identified by its unit name, a part within that unit, 'design', 'source code', 'unit test pack' etc., and where relevant a version number. The library using roles are divided into two types: browsing and updating. A library browsing role can get a copy of any item in the library, but cannot do any updating. A library updating role is able to retrieve items from the library and subsequently send back the item to be stored in the library as a new version.

In this example the admin roles, *PM Admin* and *SWE Admin* have library browsing facilities. The task roles are library updating roles. However, they are specialised library updating roles in that they only retrieve and update one item. For example *MUTP* role which has been created for a change on unit 'scanner' will retrieve and update only the 'unit test pack' part of unit 'scanner'. Figure 2 gives an overview of this architecture of the library system roles.

In the current version of the system, the task roles will obtain the relevant item from the library during an initialisation stage, and will replace the new version just prior to the role terminating. This could be changed if it was decided that the latest working versions of components should be available to others through their Admin roles.

3.8 The Meta-process: the development of the application model

This section describes the evolution of the application from narrative requirements through to implementation and evolution. To facilitate this phase, the narrative description was transformed at a slightly higher level of abstraction into two forms: one a representation of activities in an Activity Diagram, the other to a RAD to allow a more succinct description of the overall process (on a single A4 sheet). The Activity Diagram had elsewhere been found to be a useful tool to map activity independently of role responsibility, but in this more simple case its use was as a quick reference to the requirements. The Role Activity Diagram initially incorporated knowledge of the support system, however this inclusion was subsequently judged to be an unwarranted complexity and was removed. To all intents and purposes, this initial phase was the preparation of the User Model in Figure 1.

When considering the needs of a potential software engineer user it quickly became clear that it would be necessary to include in the implementation certain further behaviour not referred to in the ISPW-6 example description; activities which would be required in any proposed enactment to allow participants to function effectively in their organisation and system environment. Such activities were listed and initially mapped to user roles: those identified in the User Model. These were in effect the requirements for the environment, the Environment Model (Figure 2)

At this point the next major transformation was to scope the identified activities into system roles, to identify precisely what would be required from the support system. Apart from PW Int considerations identified, the needs of the 'participants' had to be considered, particularly how a participant might wish to navigate among and through activities, and how a participant's organisational responsibilities might be expressed through the user interface. It appeared to be a very subjective consideration; in this development a principle of 'Least Effort' was adopted, i.e. make the work of finding a work item by navigating among activities as easy as possible. In the application only one interpretation has been included; ideally a participant ought to be able to structure activities at will (constrained only by high-level process and system considerations). It became clear that was there were two distinct types of activity: those which ought to be permanently available for the participant, and those which were contextual and which ought to have transient existence. These former were scoped into Service system roles, labelled Admin, the latter into Task system roles. The mapping of these system roles to user roles is the first (diagrammatic) phase of the Application Model 1 as a RAD.

The correct establishment of interactions between system roles is critical to the success of a PW Int application, and this need drove subsequent work. To ensure smooth development an Interaction Map was prepared showing precisely the nature and function of the minimum necessary set of interactions, and when they had to exist. This was used to write the first phase of the application model; to allow instantiation of the task roles, comprising absolute minimum of functionality, in correct sequence and with correct links in place.

Subsequent development of the Application Model 1 was concerned with specialising these class definitions incorporating the information contained in the Organisation, Product and User models, and in extending the fine grained functionality of the task roles. There is wide freedom in PML to define the guards for actions to implement behaviour via the intra-role scheduler (part of the PCM), however this freedom means that guards become very complex very quickly, and even in this simple model it was not easy to implement desired changes to some behaviours.

One portion of the meta-process was implemented in PW Int as the first specialisation of the system *Base* role. This provides a number of functions: to bootstrap the application, to provide a harness within which the application can be executed, to provide services that can usefully exist at a higher level than the application itself, and, most importantly, to provide that portion of the feedback mechanism which allows external change of the application model. One can imagine the cycle of monitoring of process performance, taking a view on the changes needed to effect improvements, and effecting such improvements by dynamically changing either class definitions, enacting instantiations, or both. This process component was modelled as Application Model 2 (Figure 4).

The development of the *Library* service role was a significant effort done in parallel with the task role development in the application model. This service was implemented as a specialisation of the basic application model: the library class definitions were introduced and task roles subclassed to incorporate functionality allowing access to the library service.

3.9 Enacting the process

There are three separate contexts for instantiation of the different components of this implementation. All of them are 'dynamic' [5]. The first comprises the application models 1 and 2, loaded into the basic PW Int persistent store containing initially an instance of class *Base* role. During the execution of the process itself, a userAgent system role and Admin system role (either *PM Admin* or *SWE Admin*) are created for each participant brought within the scope of the system. Lastly, an assignment of a particular CCB change (through invocation of the action Assign Unit Change Tasks, see Figure 5) creates specific instances of task roles bound to selected participants.

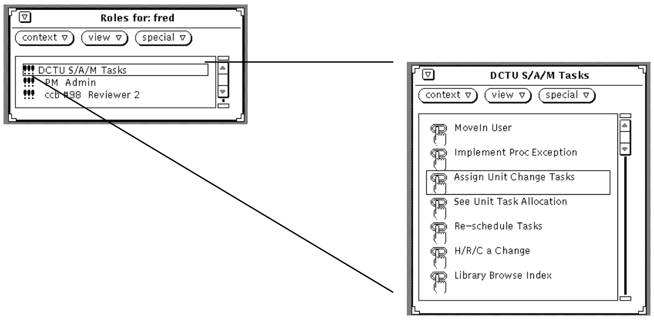

Fig. 5. UI Representation

In the PW Int support system, all implementations use a specific persistent store containing an initial specialisation of system role *Role* as *Base* incorporating functionality to change itself. This facility is used to load the code for both application models. At the conclusion of this loading, the library service is started automatically and an action in *Base* offers the action 'Run DCTU Process' and, on its selection, a prompt requests the user name of PM and proceeds to instantiate one *DCTU S/A/M Tasks* role (Schedule/Assign/Monitor), and one *PMAdmin* role, both bound to PM's user name. Figure 5 illustrates a typical role agenda for a PM whose username is 'fred', and an action agenda for *DCTU S/A/M Tasks*. In the example shown, it can be seen that fred is also the second reviewer for change reference **ccb # 98**. This bootstrap activity can be followed quite clearly on the role activity diagram Figure 4.

An essential precursor to the demonstration is for PM to 'Move In' a number of users, identified as QAE or DE. The ISPW-6 process enactment starts by PM selecting 'Assign Unit Change Tasks'. This allows identification of unit to be changed, choice of DE, QAE, and Reviewers for that particular change.

The implemented process was enacted using ProcessWise Integrator 1.1, the Process Control Manager component available both in Unix on a SUN server, and in ICL VME [14] on an Estriel mainframe located at Manchester. The user interfaces ran on SUN workstations, usually at Manchester but not infrequently at Southampton, Bristol, and Durham using the X400 protocol on the JANET network.

The system supporting the ISPW-6 example process normally runs continuously, being stopped only once per week at a convenient time for a brief garbage collection. Thus it is always available for demonstration with access from Manchester or from elsewhere.

4 Lessons Learned

4.1 Scheduling

This is a prime concern of a typical manager and careful thought was given to the extent to which it ought to be supported in the implementation. 'Scheduling' means the preparation of a plan to accomplish a piece of work in a certain time period. In order that this can be done, the manager needs to know the sequence in which discrete activities can occur, whether or not concurrency can be exploited, other constraints on the activities, and lastly, what human and machine and other resources (such as office space) are available.

Resources were not available to provide any degree of detailed implementation to support this kind of managerial activity. Many excellent software packages exist which can do this and the facility of ProcessWise Integrator to integrate a tool would be the normal way of incorporating the facility. The place for PW Int in this context is in the linking of knowledge of current activity state through 'Monitoring' to 'Scheduling' and 'Rescheduling'. In a normal design environment common obligation on designers is to complete a daily time sheet as an indication of how their time was allocated between workpackages. It would seem sensible to allow this information to be collated automatically and used for again automatic updating of records of work done and work remaining to be done in a scheduling tool. PW Int provides the necessary envelope within which this kind of activity could sensibly be provided.

Other task related information can be abstracted, e.g. when activities have been commenced, However it is not easy to abstract this information from the system in a sensible and reliable way (the system has no understanding of the data) , and recourse would likely be made to participant-provided 'time sheets'. The manager then has the current information needed: how much has been done; what has finished; when current tasks likely to finish; what should have started and hasn't.

4.2 User interface

It is necessary to visualise just how a user might want to navigate among tasks and services available. With PW Int as a support enactment system we have the opportunity to 'parcel' the behaviour with an entity and refer to it as a system role, or to provide a system role with appropriate 'tools' for holding and for manipulating entities, or some combination of the two. Criteria for choice would include just how routine the activity is, what to see if there is no work to do, volume of products, preference of users.

Certain activities were context-free, for example certain library access functions; others, such as tasks, were very context-dependent. Relevant here for example was what the screen ought to display when there were no process tasks to perform.

In addition, of course the constraints of the PW Int user interface restrict opportunity. Screen representation can be very much an area of individual preference and ideally therefore ought to have defaults accessible by the user. However as the architecture of the application model is heavily dependent on this choice, certain decisions had to be made and closed at an early stage. One of the more obvious examples of this is the location of activities. It may not be at all clear to the process designer which role is the most appropriate for a particular activity, but the choice cannot be left to the user because it may not be an easy matter to move an action between system roles, particularly if performance is an issue.

4.3 Process control

The organisation chart defines who controls which roles. Thus it is clear that PM controls the activities of Software engineers. What does this entail? Essentially it is the ability to allocate resources and to assign tasks, and the responsibility to ensure tasks are completed according to schedule. However it also includes the ability to make changes in the process. The essence of improved process performance is a clear identification of the mechanism whereby the process can be changed. This implies the existence of another 'orthogonal' process: not concerned with the day-to-day activities or performance, but the performance of the process as a whole. A process component has been written in PML called the Process Model for Management Support [15]. It provides a context for delegating objectives, planning, implementing, monitoring and re-planning process models and will be incorporated into the implementation in future.

A well-known feature of PW Int is the ability to make change 'on the fly'. This is modelled in this example process so far in only one way: to modify an existing instance of Design Review to bypass reviewers. It is accomplished by the PM selecting the appropriate action which offers a choice of CCB Changes to target, to select the appropriate modification, and to identify the target system role in which the modification is to be implemented. Other aspects of the ISPW-7 'Change' extensions will eventually be incorporated.

However behaviour changes usually involve action sequencing, and difficulty was encountered in determining process flow from action guard definitions in PML. This could be problematic in live enactments and some formalism is needed to guide the determination and testing of guard statements.

4.4 Guidance and enforcement

The paradigm of PW Int is essentially one of prescription and the work available is defined in the system and the facilities available are prescribed in the system.

It attempts to support the behaviour of the software engineer rather than merely indicate what task has to be done next. However just how much freedom participants are given is ultimately the choice of the process owner.

This will be an ongoing debate and we have yet to understand if one way is better than another, or under what circumstances one is better than another. Is automating of procedures adequate? Taylorism co-ordinated the work of craftsmen and re-structured the activities to improve productivity. It is not yet possible to co-ordinate the intellectual effort of knowledge workers in the same way.

4.5 Co-operative work

PW Int does not at present support multi-user access to system roles and thus there is no means of representing joint responsibility. Group responsibility can only be represented by decomposing the process into more elementary activities which can be assigned as roles to single users in a 1:1 relationship. In this model such difficulties were experienced in 'Review Design' and 'Test Unit', which are both joint responsibilities. To support the notion of joint responsibility a separate role would have to be created to allow identification of decisions and concurrence with or rejection of them, together with a framework for negotiation.

It is not yet possible to adequately represent the mapping of roles in organisations between groups and individuals. Singh [12] has recently made progress and uses the term role sets to represent group responsibility however they are difficult to manipulate and do not yet adequately cope with the dynamics of responsibility transfer, delegation and subsumption.

4.6 Knowledge of state

Difficulty was found in communicating to the participants knowledge of current system state, in the sense 'what can happen next and why isn't it happening'. A device was adopted through the use of dummy actions to convey this information, but a suitable mailserver to support communication in the context of the process would be useful.

4.7 Meta-model and Methodology

The work of developing these models has made a small contribution to our knowledge of how to go about implementing a particular (simple!) process description. Although PADM is primarily concerned with mapping and interpreting real-world processes, it played a part in addressing many user-centred issues, particularly those around choice of a suitable system roles and working environment. In addition the diagramming tools were useful.

It must be borne in mind that this example process is in many ways quite unlike real-world processes in the clear definition provided, the absence of cultural influences and absence of any need to consider the transition to the supported process. It also may represent only a tiny portion of the spectrum of process activity which might be considered for process support.

The ISPW-6 example is coarse-grained and represents a typical description of 'what ought to be' rather than 'what is'. There is no reference in the PM1 stage of the meta-process to establishing the 'what is' process as a basis for planning the transition to the 'what ought to be' process, but, as White mentions [16], in the real world this transition may well be the biggest problem to be dealt with.

This issue is very pertinent when considering the impacts of context and skill. The activities that are performed and the way that they are performed are frequently heavily dependent on the nature of the information to be handled, and on the skill and experience of the individual participants.

5 Conclusions and future work

The process example may well be a useful vehicle with which to appraise alternative technologies. It encompasses most kinds of purposeful behaviour and only one kind of process seems to have been omitted: that of a management control. Together with the extensions, there appears to be adequate scope for investigation into the strengths and shortcomings of technologies.

As well as offering a demonstrator for critical appraisal, this work also contributed to PADM as an example of a modelling and instantiation route into this specific enactment technology. More implementation work in Processwise and other systems will enable some guidance to be provided for those seeking to implement modelled processes.

Mention has been made (Section 4.3) of the difficulty of scheduling activity which is even of only moderate complexity. One possible solution to this problem lies in the use of OBM (Organisational Base Modelling) [11] to rigorously specify a process prior to implementation. OBM's use of operation patterns to express the often complex logic of activity behaviour may lead to a relatively simple method of defining guard statements of actions in PML. If this is successful then the way is clear to make use of requirements validation through process simulation.

It is intended that development of the implementation described herein should continue as resources permit, to eventually address in full the issues raised in the extension to ISPW-6. In particular, the integration issue has to be brought to the forefront. It may be opportune to model the example process as a pattern of constrained invocations of tools rather than making use of the rather primitive graphics of the User Interface. This may well be most appropriate application domain of PW Integrator technology. In addition, we feel that the instantiation of a coherent 'managing' process, to support planning, implementation, monitoring and re-planning, would significantly extend the scope and interest of the example.

One concern is that, as a demonstrator, the full process example is already quite complex and we anticipate different versions of the implementation to focus on different specific issues.

A notable omission from the process example, understandable in one of such simplicity, is consideration of how the process might change depending on the kinds of modules being changed. There is concern in software engineering circles that 'context' of the process activity, or the products being developed or being produced by the activity, may well have a profound influence on the process under which they are produced. A SERC funded project 'Evolution of Large Software Systems' will start in Manchester and Southampton in late 1993, investigating the means to control the evolution of systems within the context of a system architecture definition and architectural process.

The work bearing fruit in PADM will continue in another proposed SERC funded project 'Process Engineering Framework' of which the Methodology component will complement and extend earlier work under the IOPT project. Of particular interest will be the development of process theory, process taxonomies, and process engineering practice and techniques.

It is of particular interest for us to understand the extent to which the ISPW-6 process may be considered generic. There is a plan to model real world software processes using PADM to determine the extent to which they may be 'subclasses' of the process example, and if so, how specialisation might be brought about.

Ongoing action research with industry will continue to be a valuable source of real-world knowledge and a proving ground for ideas, theories and tools.

6 Acknowledgements

I would like to thank Mark Greenwood of Southampton University for his perceptive ideas and criticism and also for his substantial direct contribution to this implementation.

References

1. R.F. Bruynoghe, J.M. Parker, and J.S. Rowles. PSS:A system for Process Enactment. In *Proceedings of the first International Conference on the Software Process, Manufacturing Complex Systems.* IEEE Computer Society Press, 1991.
2. R. Conradi, C. Fernström, A. Fuggetta, and R. Snowdon. Towards a Reference Framework for Process Concepts. In *Proceedings EWSPT'92*, volume 635 of *Lecture Notes in Computer Science.* Springer Verlag, 1992.
3. R. Conradi and C. Liu. Process Modelling Paradigms: An Evaluation. In *Proceedings of the First European Workshop on Software Process Modeling*, Milan, 1991.
4. R. Curtis, M. I. Kellner, and J. Over. Process Modelling. *Comms. of A. C. M.*, 35(9), September 1992.
5. P.H. Feiler and W.S. Humphrey. Software Process Development And Enactment: Concepts And Definitions. In *Proceedings of the Second International Conference on the Software Process.* IEEE Computer Society Press, 1993.
6. A.W. Holt, H.R. Ramsey, and J.D. Grimes. Coordination System Technology as the Basis for a Programming Environment. *Electrical Communication*, 57(4), 1983.
7. M. Kellner, P. Feiler, A. Finkelstein, T. Katayama, L. Osterweil, M. Penedo, and H.D.Rombach. ISPW-6 Software Process Example. In *Proceedings of the First International Conference on Software Process*, Washington, 1991. IEEE Computer Society Press.
8. M.A. Ould and C. Roberts. Defining Formal Models of the Software Development Process. In Pearl Brereton, editor, *Software Engineering Environments.* Ellis Horwood, 1987.
9. C. Roberts. Describing and Acting Process Models with PML. In *Proceedings of 4th International Software Process Workshop*, Moretonhampstead, 1988.

10. C. Roberts and A. Jones. Dynamics of Process Models in PML. In *Proceedings of 5th International Software Process Workshop*, Kennebunkport, 1989.
11. J. Sa, B.C. Warboys, and J.A. Keane. OBM: A Specification Method for Modelling Organisational Process. In *Proceedings of the Workshop on Constraint Processing at the International Congress on Computer Systems and Applied Mathematics (CSAM'93)*, St. Petersburg, 1993.
12. B. Singh. Interconnected Roles (IR): A Coordination Model. Technical Report CT-084-92, MCC, 1992.
13. R.N. Taylor et al. Foundations for the Arcadia Environment Architecture. In Peter B. Henderson, editor, *Proceedings of 3d ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments.* In SIGSOFT Software Engineering Notes 13(5) pages 1–13 November 1988, ACM SIGPLAN Notices 24(2) February 1989.
14. B.C. Warboys. VME nodal architecture: a model for the realisation of a distributed system concept. *ICL Technical Journal*, 4(3), 1985.
15. B.C. Warboys. The IPSE 2.5 Project: Process Modelling as a basis for a support environment. In *Proceedings of the First International Conference on Software Development, Environments, and Factories*, Berlin, 1989.
16. P. White. Report on the Process Analysis and Design Methodology. Technical Report 142, IOPT, 1993.