A Support Framework for Dynamic Organizations

Mark Greenwood, Ian Robertson, Brian Warboys

Informatics Process Group,
Department of Computer Science,
University of Manchester, Manchester M13 9PL, U.K.
{markg, ir, brian}@cs.man.ac.uk

Abstract. This paper describes a framework that is intended to provide an infrastructure for the delivery of support for business processes – including design processes, software development processes and procurement processes. This framework has to be capable of supporting quasi-independent process model instance evolution, and to provide seamless support in this environment for the non-expert Process Manager to exploit this capability

1 Introduction

Many organizations see benefit in, and will provide resources for, the modelling and analysis of their processes. However very few see that benefit extending to the use of software process technology to support and enact these selfsame processes. This state of affairs is in marked contrast to that of the workflow community, which is characterized by a number of application-driven initiatives and relatively widespread adoption. There may be a number of reasons for this: greater interest by tool manufacturers, the focus on simpler processes, or the availability of "better" technologies or modelling formalisms. As yet there is no clear consensus regarding the issues on which research should focus in order to remedy the situation and the questions remain open.

The core justification for this work is the belief that process technologies, to be successful, must be capable of supporting the inherent complexity of human-intensive processes. The kinds of complexities that affect support technologies are: inconsistency in views, incomplete views, and the ongoing state of flux. This work brings into focus the need to support organizational processes to a greater extent and to a better degree than has hitherto been seen as necessary – complexity in structure and behaviour (arising from evolution) has not yet been addressed by contemporary modelling and enactment systems.

The approach described in this paper is based on the concept of structuring software development processes in particular, and business processes in general, in a manner that is analogous to the way in which systems (software systems and others) are designed and implemented in the real world. It aims to support and exploit the diversity and richness of the different methods used by organizations to achieve their goals, yet be able to accommodate the dynamic nature of such organizations.

The following section describes the context of this work in the PIE project, with related work in section 3. A key aspect of this project is its emphasis on evolution, so there follows in section 4 a brief discussion of relevant evolution issues. Section 5 describes the framework that is proposed as the basis for addressing certain core problems identified for the project. Section 6 provides an overview of the Process*Web* technology in which the framework is currently implemented. An example application of the framework is briefly described in section 7, and conclusions are expressed in Section 8.

2 Context

This paper describes the aims of the Evolution Strategy (ES) workpackage of the Esprit PIE[1] project together with the architecture and technology which, it is believed, will be capable of contributing significantly to all of the major results of the project but in particular to a) the framework for process evolution, and to b) the development of tools to support definition, execution, and dynamic modelling of organization processes [7].

Evolution Strategy is one of the services needed to bring the goal of the project within reach. Its focus is on the organization and its processes, and it acts as a mediator between the PIE Middleware infrastructure to be provided by other workpackages [6] and the services available from the components of Monitoring and Decision Support (MS and DS respectively) [1], and ultimately also a Change Support (CS) component. From another viewpoint it is acting as a mediator between the user and the technologies available to the user via the PIE Middleware.

It operates at the same level as these components because, although certain evolution in an organization can be imposed top-down, there are many coordinations and adaptations that are developed and implemented at local levels in contexts that are quite isolated from high level concerns. Therefore Evolution Strategy is viewed as a component just as the other services are components, and as other process support systems are components recognized by the PIE infrastructure.

In undertaking this mediation role, the ES component has some specific aims which are laid down in the objectives of the project, and enhanced where appropriate by the emerging requirements being determined by the project's industrial partners. The results are intended to allow a non-specialist Process Manager to:

- Treat a very complex process model, composed of numerous model components or fragments, as if it were monolithic.
- Use preferred methods for the modelling and evolution of process models, be they on-line or off-line.
- Evolve very large and complex process models, such as those for software development, in a responsive manner (involving feedback) in a compact time frame.
- Evolve them taking cognisance of the fact that their implementations in different businesses may be on different platforms, and indeed may be paper-based.
- Focus attention on particular process model instances in a concurrent fashion by means of software components that support monitoring, decision-making and change, and to be able to tailor these components to suit contextual needs at run time.

1. Process Instance Evolution (PIE), a Framework IV LTR project.

- Add models incorporating successful changes to a library, thus supporting organizational learning. These models would initially be specialized process models, and ultimately they could be generic.
- Accommodate changes being applied to the Process Manager's domain through different Evolution Strategies located elsewhere in the business (or even outwith it).

The Framework is an architecture implemented as a process model. This process model can generate a network whose nodes represent products and processes of the organization (or placeholders for these). In addition, other process models or PIE Components, referred to as methods, can be invoked through operations of the nodes to support product and process evolution. The PIE Components help identify the need for process model evolution (MS), determine alternative models and select the most appropriate(DS), and implement this model in a running process model instance (CS). They are controlled by means of a meta-process that is built into the framework.

The workpackage is split into two tasks, each covering one year. The first is concerned with developing the mediation framework, the essence of which is described in this paper, and the second is concerned with supporting concurrent evolution, i.e. when a process model instance is subject to evolution from a number of unrelated sources. Both tasks are driven by user requirements and the industrial case study prepared by the industrial partners (introduced in [1]).

3 Related Work

Supporting a business process through technology often provokes changes to that process. In part this is a direct result of the attention given to the process to decide how it ought to be supported. However, this merely accelerates the natural tendency of business processes to evolve as new requirements and opportunities emerge. Early experience with a software development environment illustrated that a process support system's long term effectiveness is restricted if it is difficult to adapt to new methods and tools [15].

The need to support processes over time is illustrated by the research on meta-processes, processes which deal with other (operational) processes [5][8]. Our implementation experience [17] is that while meta-processes support evolution there is still a complexity problem. A complex meta-process to support the evolution of a complex operational process can be very unwieldy.

This framework exploits a hierarchy to manage complexity. This is based on previous work in managing large specifications [14], and software systems [10]. The framework provides a process architecture to complement a meta-process in supporting evolution. The expectation is that many changes will not involve changing the overall system architecture, though the meta-process does understand the architecture enough to do this if required. The implementation gives a generic process system which is adapted to the business process being supported.

The notion of a process system which has the ability to adapt to new demands is not unique. The Endeavors system keeps an internal model of itself which can be used by enactable models supporting the development, installation, customizing and reconfiguring of the system [3]. Amber has experimented with a meta-linguistic approach, where callbacks to instance specific code is used to modify the system's default enactment behaviour [11]. However, most emphasis has been on configuring a process system to specific demands of one model, rather than supporting ongoing adaption over a sustained period of time. Similarly WebFlow [2] has mechanisms to support the evolution of a process model at the micro-level but does not address the broader, policy issues involved in supporting the evolution of complex processes.

4 Structuring Evolution

A common theme running through this work is the phenomenon of evolution and, before describing the framework on which the solution will be based, the issue of evolution will be expanded upon.

Evolution is a transformation. It is adaptive change with a time dimension, which implies that:

- the changed system is based on the earlier system,
- there exists a structure to maintain equilibrium between the system and it environment,
- there are issues relating to successive and concurrent system changes.

Evolution can be directed or spontaneous, that is, it can be externally imposed or internally driven. An example of directed evolution would be the shift of a business in a new direction such as retailers expanding into the banking business. An example of spontaneous evolution could be local initiatives of process improvement. Directed evolution is not always top-down, for example the propagation of a new software technology throughout an organization can be orthogonal to the normal hierarchy of line responsibility. Different kinds of evolution can coexist, so process support needs to be able to accommodate changes arising from a number of sources, possibly concurrently.

Evolution in organizations is at the same time both focused and diffuse. Much of evolutionary change is a result of purposeful strategic decision, for example to enter a new market, to change a product line, or take-over by another business. On the other hand it is diffuse in that a) strategic evolution is implemented by many minor evolutions, and b) minor evolutions can take place on a local basis that may well have no impact on business strategy. It cannot be said that evolution is entirely externally constrained, nor that it is isolated, and this leads to the notion of "quasi-independence" which will be discussed later in section 5.4.

In looking at systems that include many related process model instances, the issue of "co-evolution" has to be considered. This refers to the situation where the evolving system has a dependency on another system. The consequence is that evolving one system might result in an inconsistency between the two systems, and this inconsistency has to be addressed. This has to be either a corresponding evolution of the second system, a reversion in the first system, or a tolerance of the inconsistency. There are three common situations for process co-evolution, and this work is concerned about all three:

- Co-evolution between two related process models.

- Co-evolution between the definition domain and the enactment domain [9].
- Co-evolution between enactment and performance domains.

One consequence of recognizing the time dimension is that the relative duration and frequency of changes becomes significant. The duration of a cycle of evolution, from initial feedback to applied change, is related to the life expectancy of the process model instance and the responsiveness expected of it. The frequency with which it is possible to apply changes means that there is unlikely to be a 1:1 relationship between reported inconsistencies and applied changes, and the evolution process will need to deal with aggregation of feedback.

In order to separate out the different issues that must be addressed, and to give an indication of evolvability, we use the term "mode" to identify, where possible, distinct and orthogonal models of evolution processes. It is explained in section 5.4.

Evolution can also contribute to the maturity of an organization if the results of experiences can be incorporated into models of process behaviour that can be usefully exploited on future occasions.

5 The Framework

The framework is referred to as the Product Evolution Tower (Peltower) and consists of three core features: nodes, their structure, and evolution. The structure relates one node to another, and all features are subject to evolution.

The Peltower is generic and can be specialized in two ways: first of all, certain specific behaviours can be identified in the node, and secondly, the structure can be defined as a particular kind of hierarchy, where child nodes have a specific relationship to the parent. It has so far been implemented in two particular examples–the Hdev (Hierarchical Development) tower [10], and the OPM (Organizational Process Modelling) tower [16]. The former is used as the basis for the PIE investigations and development.

Such hierarchical structures are commonly evidenced in system design to represent the relationship between design components. In order to implement a specification (either as a design or as an artefact) it is necessary to decompose or refine the specification into manageable units, i.e. where they can be transformed into design artefacts with appropriate modularity properties, or can be directly implemented by a known process. Thus each node represents a part of the product that can be usefully encapsulated and interfaced with other nodes. The decomposition of the specification is determined by known technologies.

Although based on a software development process [10], it is believed that such a generic approach can usefully support the kind of process activity evidenced in dynamic organizations. The framework makes no claim to be able to support all aspects of activities, but it is intended that off-line coordination will be accommodated by the structure. The framework aims to support all process aspects of business activity whether the outcomes of these processes are products or services. It is envisaged that a population of Peltowers could in effect represent all the models and instances of processes and meta-processes used in some particular context such as an organization, whether it is real or virtual.

5.1 Node

A node is characterized by three properties: specification, product and type. A specification represents a solution to a problem. It is a definition expressed in terms of behaviours and technologies which, if designed and implemented as a system, and executed, would resolve the problem. The specification property in the node is a representation of this. Likewise the product may be in a variety of media and the product in the Peltower framework is intended to represent the actual product that will probably exist in some other medium. This product is not intended to be solely a data object. It can represent both data and behaviour as does a conventional software system, or essentially behaviour, as does a process model. In fact at its most general, it represents *any* result that satisfies the specification.

The use of the terms is intended to be relative rather than specific. A product in one context can be a specification in another. For example, the output (product) of a requirements analysis process is a specification for use in a development process.

As well as the specification and product properties, a node consists of a number of key behaviours that are referred to as operations. These are primitive mechanisms that utilize behaviours defined in methods. Operations, like methods, are characterized by a type signature that defines inputs and outputs and which is also used to provide resources to the method. Methods are discussed further in section 5.3.

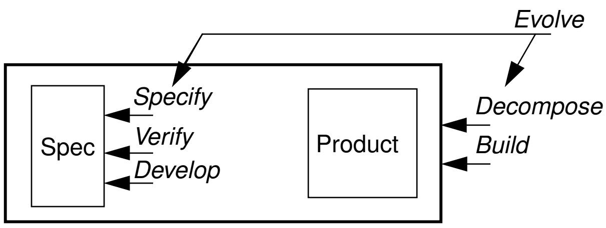

Figure 1 Hdev type of Peltower node

There several node types, and a node type is characterized by the available operations on Specification, Product and Node. As well as the Hdev and OPM types that constitute the already-mentioned Hdev and OPM towers, there is a Root type to facilitate the bootstrap of the framework in an implementation. There is also a Developer type to enact a model written in a Process Modelling Language (PML) [4] in order to develop (code) enactable models in the same language. The following operations are available in the Hdev node (see figure 1).

Specify. Operates on the specification to change it. A specification can be provided when the node is created, or it can be prepared subsequently. This operation is related to the Verify operation which is discussed in the following section. A change to a specification, and its consequent development into a new version of the product provides one of the modes of evolution discussed further in section 5.4.

Develop. This operation relates the specification to the product in any kind of system or part of a system. It can be a design process, where the product is a design artefact that can be used in turn as a specification for a manufacturing process to produce a tangible artefact, or a procurement process or a software development process. The framework does not specify the behaviour of Develop other than to indicate that it causes a transformation of the specification and results in a product. The actual behaviour that supports this activity is called a method, and it is context-sensitive. A full explanation of methods is provided in section 5.3.

Evolve. This can be applied to *all* operations except itself and, as currently defined, allows for one particular method definition to be substituted for another. Thus for example the process by which the product is developed can evolve. The method for this particular operation is loaded with the system, and it is not evolvable. This was a decision taken to constrain the complexity of the current implementation. The evolution process is discussed further in section 5.4.

The other operations of Verify, Decompose, and Build shown in figure 1 are related to a network of nodes and will be explained in the following section on structure.

5.2 Structure

It is usually impossible to develop complex artefacts such as a software system directly from a high level specification. As mentioned at the beginning of this section, an essential prerequisite for development or implementation is for the specification to be refined or decomposed into more manageable units of appropriate modularity that can be developed by an individual or a small team. The parent–sibling relationship in this decomposition is 1:N "part-of"[14]. This means that the specifications of all the siblings can be combined in such a way as to be equivalent to the parent specification, and this indicates how the child products can be combined to satisfy the original specification. The following description of the remaining node operations applies to the Hdev structure.

Decompose. This operation generates one or more child nodes of the same type as the parent and establishes certain links between them. The child inherits the parent's methods. Decompose implies some kind of refinement of the specification, and indicates co-operation between nodes. There is no limit to the number of times the Decompose operation can be invoked: if the first decomposition is incorrect, further children can be added or a child can be terminated. Terminating (thus deleting) a child has no effect on the parent or any of the other children, but terminating a parent results in all its children (and children's offspring) being terminated. The only inhibitor to termination in this case is if an operation is executing a method. In this case, termination will not proceed on the node until the method has concluded.

Verify. This is an operation that checks the consistency of specifications between a parent and a child, and is a key operation in one of the modes of evolution. It can be invoked after a decomposition when any inconsistency arising would need to be corrected in the child specification. More importantly however it would be invoked after any change in specification at any node. In figure 2 for example, a change might be made to specification of node 1.2 for some reason. After the change, the specification would be verified with respect to its parent (1) and its children (1.2.1 and 1.2.2). If the result revealed inconsistency with the children's specifications, then they would also need to be changed. With regard to the parent, if the verify revealed no inconsistency, then the remainder of the tower would remain unchanged. If inconsistency was revealed, then the change would be propagated elsewhere in the tower.

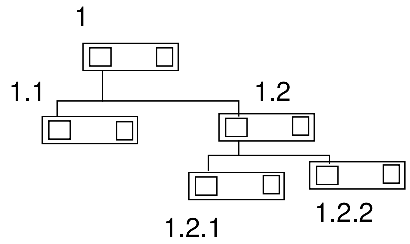

Figure 2 Peltower as a Hdev hierarchy

Build. This operates on a parent node, and combines the products of its children into a system complying with the parent specification and associating the result with the product property of the parent node. Just how it combines the child product depends on the particular method that is bound to the operation. It is, in some respect, the corollary of decompose.

5.3 Methods

A key concept in this framework is the separation of behaviour into "mechanisms" and "policies". By this we mean that mechanisms are core behaviours for some basic transformation that has to take place, and policies define a) the way that these mechanisms are used or b) provide the detailed behaviour for such mechanisms at run time [12]. The latter approach has been adopted in this work. Just how a transformation is effected is governed by the method (i.e. the policy) that is used by the operation (i.e. the mechanism). Invocation of the operation will call on a method that defines exactly how the operation will be carried out. For every operation there might be several alternative methods, one of which would be a default.

In essence, methods (and thus operations) are relatively easy to evolve at run time. This means that, for example, system developers would have at their disposal a choice of methods to carry out actual development, and they could be changed by an evolution process which at their disposal. Feedback from the success or failure of different methods could be a focus for organizational learning.

Much of the thinking underlying the Peltower framework has gone into putting as much behaviour into methods as possible, and as little into the operation mechanism as possible. In so doing it has enhanced its potential for evolution.

Methods so far defined for Peltower implementations have been simple although they may conceivably be quite complex and need to be evolvable. Thus the same principles that apply to the Peltower should also apply to the method, so logically they should be structured by another Peltower and re-used where and when needed. This thinking also extends to the other PIE components. It is intended that a method structured as a tower will be used to encapsulate such ad hoc software technologies in order to provide services needed by the users through the process model in a much more effective and context-sensitive way than has so far been possible.

5.4 Evolution

Evolutionary change can take place in a number of ways in the Peltower structure. The cyclic behaviours that bring them about are referred to as modes, and two are involved: the Specify and Verify operations together, and the P2E meta-process (they are explained later in this section).

When discussing evolution, a core question is, *what* evolves with respect to *what* [13]? Classically, the object is said to evolve with respect to its environment, and, in our context, the object is part of a software system. Peltower objects for which evolution support is provided are[1] listed below. In each case evolution is initiated by a user.

- Product. The combination of the specify and verify operations together provide for the Product mode of evolution. A change in specification results in a change in product. However, more importantly, change in specification can be ripple throughout the hierarchy, thus propagating product evolution wherever it is needed. This is quasi-independent evolution, by which we mean that the specification property of a node can evolve independently but only to the extent that its neighbours can accommodate such change.
- Operation. This is change in the behavioural definition of the particular method associated with an operation. As one of the operations is responsible for defining the structure of the tower (Decompose), it also handles structural evolution. The mode used is the meta-process known as P2E (see below). It is model evolution.
- Node. When applied to a node instance (via the P2E mode) the node instance type is changed, and thus its pattern of operations evolves.

The Specify/Verify evolution mode has already been introduced when discussing the operations that are possible on a node. The second mode is P2E, the Process for Process Evolution [8][16]. In this framework the P2E is a latent mode. By "latent" we mean that it is not instantiated until it is needed, i.e. until there is a need for operation or node evolution. When instantiated, it is parameterized for the operation whose method is to be evolved, and the node type. (This aspect will be reviewed in the light of emerging requirements of the PIE project.) It is the mode for evolving both methods and nodes.

Figure 3 shows the basic relationship between a P2E and its node. There are two interactions. One, from the node to the P2E, is "sensory", a form of feedback that provides information to the P2E about the node's type, and about its operation whose method is to be changed. The other interaction is "motor", one that imparts change. The Re-

---

1. It is envisaged that method evolution would take place in the Peltower whose products are themselves methods.

alizing component of the P2E mode applies a definitional change to the node instance so to implement the desired change and conclude the cycle of evolution.

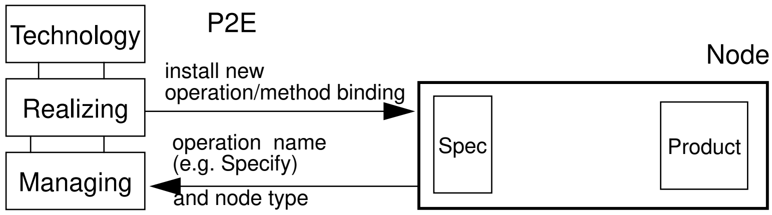

Figure 3 Evolving a node operation

The use of the P2E mode allows the support for humans participating in the evolution process. The P2E itself has operations, such as ObtainMethod for finding methods. This operation resides in the Technology component of the P2E and tries to find a class definition that matches the type specification of the particular operation. The framework recognizes that a suitable method might not be available, so it allows for a further cycle of evolution to determine a suitable method for the development of method. See figure 4.

The new P2E, on instantiation, is passed the name of ObtainMethod and the type of the parent Technology. Its own Technology finds or determines a new way to implement ObtainMethod, and installs this new definition in Technology as new method. An example of this is where we, instead of *finding* a method, want ObtainMethod to provide the means of *creating* one. The instantiation of a second cycle of evolution is a process of abstraction.

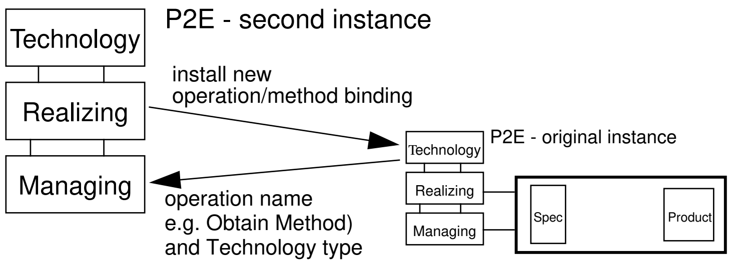

Figure 4 Evolving a P2E operation

There is nothing in the framework to inhibit concurrent evolution, however implementation details may impose constraints. In the Hdev implementation, product and operation modes allow concurrency. Others require instances of P2E to bring about the evolution, and only one instance of P2E can be associated with a node instance at any one time. There is no restriction on evolution between different node instances.

6 Technology - Process*Web*

All the PIE components - Evolution Strategy, Monitoring and Decision Support [1] - share where possible the common architecture illustrated in figure 5. This has been done in order to facilitate a) the addressing of common problems, b) a level of integration, and c) evolution. The architecture will be explained by reference to the Process*Web* [17] technology.

Process*Web* combines TeamWARE's ProcessWise Integrator (PWI) with a Web server so that users interact with the system through the familiar medium of their Web browsers. The ES component comprises an instance of Process*Web* running a Peltower framework process model, and PWI's process engine and the Peltower model are located in the Kernel of the architecture.

The GUI layer is a web browser. The Process*Web* system calculates and assembles html pages as required by the execution of process model instances. These pages include links or standard Web CGI (Common Gateway Interface) forms, through which users provide input. This user interface communicates through the PWI external application interfaces (API in the architecture). Thus ES departs slightly from the common architecture which proposes a direct connection between Kernel and GUI layer.

The same PWI application interface allows communication with databases or other software tools at the "global" Process*Web* level, and in the context of the PIE project, for communicating via a Communications Layer with the PIE Middleware.

The Persistent Layer for Evolution Strategy is conceptual. This is because the language used to program Process*Web* is persistent – there are no explicit commands to write or read data from a persistent file or database. The PML language implementation ensures that a consistent state is regularly written to a persistent store, and also manages reading data from the store to memory as required.

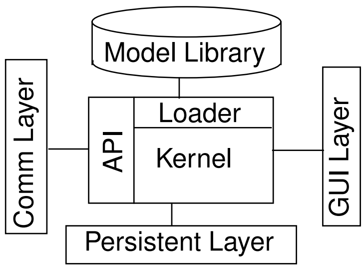

Figure 5 The PIE Common Component Architecture

The Loader makes use of a PML primitive which allows run-time compilation of PML code. This allows the methods which are located conceptually in the Model Library to be evolved independently of the Peltower implementation.

Basic constructs in PML include a Role (a thread of behaviour with its own local data), and Interaction (an asynchronous buffer which is used to communicate messages between roles). PML is a class-based language with single inheritance and incremental compilation, where a source PML class definition is compiled in the context of a set of existing classes.

An Hdev node is implemented as a network of PML roles and interactions. The key Hdev role includes the specification, the product, and the name, type and class definition of each operation, within its local data. The behaviour part of the Hdev role can express any constraints on the operations, e.g. that the develop operation cannot happen if there is no specification. There is a corresponding HdevUI role which deals with the presentation of the current state of the node. It has interactions with a UserRole which is a standard Process*Web* role for interfacing with a user. This design provides a clearer separation between the coding for the Hdev node and the coding for the user interaction. For example, two HdevUI roles could be used to enable two users to work concurrently on a node, conflicts between them being handled by the Hdev node.

The methods are also implemented as PML roles and contained in the Kernel model, however conceptually they are maintained in a Model Library. This is another example of the approach that separates policies from mechanisms (see section 5.3). When an operation is invoked, a role of the appropriate class ('"loaded" from the type space) is created and initialised with the operations input parameter. The method role is also given interactions which enable it to communicate with the UserRole, and hence the user, and also an interaction which it uses to return its result to the node which created it. Implementing methods as PML roles enables the Hdev tower implementation to use the PWI error handling facilities to inform the node if there is an error in the method code. It also means that more complicated methods can be implemented as networks of roles and interactions.

The P2E is similarly implemented as a network of roles and interactions. A P2E connected to a node (see figure 3) may be created in response to a request to evolve an operation, a request to evolve the node type, or because an existing method has failed reporting an error. Once the source PML class definition for the new method has been found, using ObtainMethod, then this definition is compiled to check for errors and consistency with the other PML class definitions which the node is using. Finally the new method class definition is installed in the node, updating its local data, updating its local data for the appropriate operation. Where a node type is evolved, there may be several new methods to obtain. In this case the implementation creates a new node, and transfers to it the specification, product, and the interactions which are its connections with other nodes in the hierarchy.

7 Example

This framework is being implemented in the context of the "Hood and Headlights" scenario that is being defined in the PIE project, and which has been introduced in [1]. The context is set by the Peltower being populated by motor car specifications, designs and design process instances. For example, the specification for a car body can be decomposed into a number of specifications that relate to the body shell, doors, headlights and hood etc. The body is represented by a node, and the sub-components by children of that node. Some of the process model instances will be running in another process support system, and the Peltower representation will then be a placeholder allowing communication with the actual instance.

Each node contains a process for developing its specification into a design. They are usually design processes, but equally they may be procurement processes. They are all supported by process technologies. The scenario envisages the situation where designs for the hood and the headlights are found, on an integration test, to be in conflict, and this conflict cannot be handled by that activity. The associated P2E instance then supports the determination of a conflict resolution process (involving new instances) and integrates the other PIE components where appropriate. On conclusion of that process, the integration test can be resumed.

8 Conclusions

8.1 Managing Workflow Applications

One consequence of the success of workflow technologies has been a proliferation of applications without any means of supporting their systematic evolution. Until now, this has not been seen as a serious problem although workers are attempting to address the evolution issue on a technology-specific basis. This trend, together with the natural desire to support more complex and longer-lived processes, will expose the shortcomings in evolution support. This work has shown that such failings can be overcome by use of software process technologies supporting a framework such as is described in this work together with the other Components being developed in the PIE project.

8.2 Managing the Evolution Process

We are now beginning to appreciate the true nature of concurrency in evolution cycles, and the likely need to be able to support the aggregation of feedback and consolidation of changes.

A "case-based" approach might be appropriate to manage a number of potential change cases within an evolution process. This would allow progression of potential changes at different rates, and for the combination of prospective changes at any time up to the point of installation of the change.

Two modes have been identified for handling evolution in the framework. It will be interesting to see if more are required, or, conversely, if Specify/Verify is simply a particular specialization of P2E.

8.3 The PIE Project

The advantage of handling evolution through P2E is that it is itself based an intuitive process and is therefore appropriate to supporting the non-specialist. The results from the earlier first phase of PIE confirmed the appropriateness of this approach.

The PIE approach endeavours to consider models, tools, applications, and support systems intended to support human activity in a more consistent way, and an interesting aspect of the project is to blur the distinction between process model and tools. Their common purpose is support for organizational process-based activities, but to do this there needs to be integration at the appropriate level. This closer integration with software components may lead to more effective support for process workers, i.e. to bring tools, process and participant behaviour together seamlessly

In the same way the Peltower framework reflects this philosophy, in that process models and applications are simply methods for dealing with problems, and problem-solving is a core activity for managers and professionals everywhere in industry.

8.4 The Peltower Framework

The Peltower framework provides a promising approach to better address the true complexity of the real organizational behaviour. It does this by establishing a simple relationship between different process models that cooperate to define organizational behaviour. In addition, it provides a proper context for process evolution, recognizing that evolution is multi-faceted and this must be understood if support for evolution is to be a realistic aim.

It is accepted that most organizations are hierarchical in the way that they are set up in terms of responsibility, but evidence has yet to accumulate to justify our assertion that the framework can usefully represent the aggregation of process behaviour in *any* context.

Operational processes are not simply procedural: there is no reason why they should not be capable of changing and evolving in certain constrained ways, if such evolution does not affect another process. If it does, then the Peltower structure needs to be used to handle the co-evolution.

The potential relationship between Peltower hierarchies can be complex. Reference has already been made to the use of the Peltower to structure methods, and that these methods can largely be software systems. Other hierarchies are conceivable, for example an Abstraction/Specialization tower relating an abstract or generic process model to specialized and instantiated models (also known as template–customized–enacting).

Modelling techniques tend to assume that a process exists in isolation. The Peltower framework offers the possibility of a more comprehensive support to dynamic process activity.

References

[1] I. Alloui, S. Beydeda, S. Cîmpan, V. Gruhn, F. Oquendo and C. Schneider. "Advanced Services for Process Evolution: Monitoring and Decision Support." In R. Conradi ed. *Proceedings 7th. European Workshop on Software Process Technology EWSPT'00*, Kaprun, Austria, 2000.

[2] J.-M. Andreroli, C. Fernström and J.-L Meunier. "A Coordination System Approach to Software Workflow Process Evolution." *Proceedings of the 13th. International Software Engineering Conference ASE'98*, Honolulu, USA, 1998.

[3] G. Bolcer and R. Taylor. "Endeavors: A Process System Integration Infrastructure." *Proceedings of the Fourth International Conference on the Software Process*, Brighton, UK, 1996.

[4] R. Bruynooghe, R. Greenwood, I. Robertson, J. Sa, R. Snowdon, and B. Warboys. "PADM: Towards a Total Process Modelling System." In A. Finklestein, B. Kramer and B. Nuseibeh, eds. *Software Process Modelling and Technology,* Research Studies Press, Taunton, UK., 1994, pp. 293-334.

[5] R. Conradi, C. Fernström, and A. Fuggetta. "Concepts for evolving software processes" In A. Finklestein, B. Kramer and B. Nuseibeh, eds. *Software Process Modelling and Technology,* Research Studies Press, Taunton, UK., 1994.

[6] G. Cugola, P. -Y. Cunin, S. Dami, J. Estublier, A. Fuggetta, F. Pacull, M. Rivière and H. Verjus. "Support for Software Federations: The PIE Platform." In R. Conradi ed. *Proceedings 7th. European Workshop on Software Process Technology EWSPT'00*, Kaprun, Austria, 2000.

[7] P.-Y. Cunin. "The PIE Project: an Introduction." In R. Conradi ed. *Proceedings 7th. European Workshop on Software Process Technology EWSPT'00*, Kaprun, Austria 2000.

[8] J.-C. Derniame, A. Kaba and D. Wastell, eds. *Software Process: Principles, Methodology, Technology,* Lecture Notes in Computer Science, Vol. 1500, Springer Verlag 1999, pp. 53-93.

[9] M. Dowson and C. Fernström. "Towards Requirements for Enactment Mechanisms." In B. Warboys ed. *Proceedings 3d. European Workshop on Software Process Technology EWSPT'94*, Villard de Lans, France 1994.

[10] R. M. Greenwood, B.C Warboys and J. Sa. "Cooperating Evolving Components - a rigorous approach to evolving large software systems." *Proceedings 18th. International Conference on Software Engineering*, 1996.

[11] G. Kaiser, I. Ben-Shaul and S. Popovich. "A Metalinguistic Approach to Process Enactment." *Proceedings of the Fourth International Conference on the Software Process*, Brighton, UK, 1996.

[12] R. Morrison, D. Balasubramaniam, M. Greenwood, G. Kirby, K. Mayes, D. Munro and B. Warboys. "A Compliant Persistent Architecture." *Software Practice and Experience.* To appear.

[13] I. Robertson. "Evolution in Perspective." *International Workshop on Software Evolution*, Kyoto, 1998.

[14] J. Sa, B. Warboys and J. Keane. "OBM: A Specification Method for Modelling Organisational Process." *Proceedings of the Workshop on Constraint Processing CSAM'93*, St Petersburg, 1993.

[15] B.C. Warboys. "The IPSE 2.5 project: Process modelling as the basis for a support environment." *Proceedings of the First International Conference on Software Development, Environments and Factories,* Berlin, Germany, 1989.

[16] B.Warboys, P. Kawalek, I. Robertson and M. Greenwood. *Business Information Systems: a process approach,* McGraw-Hill, London 1999, pp. 37-54 and 182-198.

[17] B.S. Yeomans. *A Process-Based Environment for the Evolutionary Development of Large Software Systems*, M.Res Thesis, University of Manchester, 1997.