# Process Modelling Languages: One or Many?

Reidar Conradi*, Norwegian Institute of Technology, Trondheim, Norway
Chunnian Liu, Beijing Polytechnic University, Beijing, P.R. China

September 19, 1999 — Submitted for EWSPT'95, April 1995, Leiden

**Abstract**

The paper describes the different phases and subdomains of process modelling and their needs for conceptual and linguistic support, and in what forms. We group the relevant factors into three dimensions: meta-process phases, process elements, and the tool/user views. In the first dimension, we focus on enactable process models. For such models, we describe the design alternatives for a core process modelling language and a set of tailored sub-languages to cover special process elements. However, no detailed and functional comparison of possible modelling language are attempted.

Then we address interoperability between related sub-models and its implication to the language design. We also present a general architecture for a Process-Centered Software Engineering Environment, with a segmented repository of model servers.

Some concrete language realisations, mainly from the EPOS PSEE, are used throughout the presentation. We also give a realistic example of the design of an interoperable PSEE, and discuss how it can be improved using an extended EPOS.

The paper concludes that we have to live with many sub-languages around a core process modelling language. However, the underlying linguistic paradigm in this core language is not judged critical. What counts is use of standard support technologies, interoperability to handle heterogeneous and distributed process information, non-intrusive process support, end-user comprehension, and flexible support for evolution (meta-process).

## 1 Introduction

In the last decade, there has been much interest in the *software process* as a vehicle to improve software production. Many *Process-Centred Software Engineering Environments (PSEEs)* have been constructed and documented, although to a less extent tested in industrial settings. A few commercial PSEEs have also become available in the last years, e.g. Process Weaver from Cap [Fer93], and Process Wise Integrator (PWI) from ICL [Rob94].

The goal of this paper is to pragmatically discuss the design of the "ideal" *process modelling language* versus the functionality required. A special concern is how existing modelling concepts and linguistic constructs can be used and combined. Therefore, interoperability between different languages and related models and tools are emphasised. Some aspects of a general PSEE architecture are also discussed.

Discussing the "right" modelling concepts and language(s) for a certain application domain is very ambitious, and the actual design of language can be hard to validate ("air castles"). We

* Dept. of Computer Systems and Telematics, Norwegian Institute of Technology (NTH), N-7034 Trondheim, Norway. Phone: +47 73 593444, Fax: +47 73 594466, Email: conradi@idt.unit.no.

are faced with similar design decisions in Information Systems Engineering, Business Process Reengineering, Enterprise Modelling etc. – resulting in a plethora of modelling formalisms. See also [LSS94].

For instance, we may choose to have one large modelling language for the entire domain. Another alternative is to have one small and general core language, with a spectrum of specialised sub-languages to cover different subdomains (activities, artifacts etc.), to cover different goals (understandability, executability etc.), or to be oriented against different user roles (expert designers vs. normal users). We should also pay attention to standardisation and new platform technologies, cf. distributed architectures and workflow systems.

To obtain more solid assessments and comparisons of the modelling alternatives, we should have a well-defined set of domain subcategories, i.e. process elements. We will therefore focus on process modelling languages for rather "low-level" or enactable process models, not on the more high-level modelling languages to analyse and design more abstract process models.

There is yet little agreement on basic concepts, terminology, linguistic constructs, and tool architectures underlying such PSEEs. Some classification of concepts can be found in [CFFS92] [Lon93] [FH93] [CFF94], of languages in [CLJ91] [ABGM92], and of architectures in [BBFL94]. We will use the below definitions.

By *process* we understand a set of goal-oriented, partially ordered and interacting *activities*, and their associated *artifacts* and resources (humans, tools, time). Activities, artifacts, tools etc. are called *process elements*. We will limit ourselves to *software processes*, covering both development and maintenance of software. A process consists of a *production process*, a *meta-process* to regulate evolution of the whole process, and a *process support* (the PSEE) consisting of a process model and various process tools (Process Engine etc.). The production process is partly *external* to a computer, and is carried out by humans assisted by computerised production tools, e.g. CASE tools. The process support is normally *internal* to the computer, but may be supplemented by manual procedures. A PSEE has an explicit definition of the process, a *process model*. The model is expressed by one or several *Process Modelling Languages (PMLs)*, perhaps at various levels of abstraction or granularity (from template to enacting), and/or covering different parts of the process. It is usually stored in a PSEE repository, and consists of *model fragments*. The model can be interpreted to provide the user with process assistance of some sort (guidance, control, automation).

The paper will specially draw on examples from the EPOS PSEE [JC93] [C$^+$94] and its process modelling language called SPELL. EPOS has been developed at NTH in Trondheim since 1989, first as part of a national project, later as a Ph.D. project. The emphasis is flexible and evolving process support during enactment for multiple software developers, using conventional development tools and working on files in checked-out workspaces. EPOS process models (task networks) are expressed in SPELL, an object-oriented, concurrent and reflective modelling language. The process models are stored in EPOSDB, a versioned software engineering repository. EPOSDB supports nested and cooperating transactions coupled to projects, and there is one Process Engine per transaction. The underlying platform is Sun workstations running Unix, and Prolog is the SPELL implementation language.

The paper is organised as follows. Chapter 1 is the above introduction. Chapter 2 discusses the process modelling subdomains, the PML design alternatives, and some solutions in EPOS and a few other PSEEs. Chapter 3 discusses the design of PMLs in an interoperability perspective, and presents a general PSEE architecture and an example to illustrate this. Chapter 4 contains a conclusion.

# 2 PMLs: Functionality versus Solutions

In this section, we will discuss the different design alternatives for PMLs, both generally and wrt. to the functionalities that must be covered. The EPOS PML(s) are used as an example, but with comparisons to PMLs in other PSEEs.

## 2.1 The PML Design Dilemma

Much research effort has been spent on designing the "right" PML. We can identify four approaches **L1–L4** for PML design, and all include a **core** PML, cf. [ACF94] :

- **L1: One fixed and large core PML**.
  Here, the core PML contains language primitives to express all relevant process elements. Although Occam's razor might be successfully used, a large but common PML will result.

  Typical examples of "hard" language primitives include constructs for activity triggering and concurrency.

- **L2: One extensible and smaller core PML**.
  Here, the core PML contains less primitives, rather a set of declarative constructs. Thus we can define tailored process models (often types and their instances), still within a common PML.

  Examples are "soft" descriptions of product structures, user roles, or tool interfaces.

- **L3: one core and several compatible sub-PMLs**.
  Here, many of the above process elements will be covered by separate and usually more *high-level* sub-PMLs. These may have well-defined interfaces to or be down-translatable to the core PML. We could also envisage an inverse translation from a more high-level core PML to alternative low-level sub-PMLs, e.g. to generate alternative implementations; cf. last comments in Section 2.2 and most of Section 3.

  Examples are descriptions of check-out/in of workspaces, metrics collection, or transaction protocols.

- **L4: one core and several incompatible sub-PMLs**.
  Here, such sub-PMLs will be separate languages, but wholly independent of and often orthogonal to the core PML.

  Example are descriptions of versioning or user interface paradigms.

The chosen strategy for PML design will influence the size and complexity of the PML and the resulting PSEE. A good language design assumes that we understand the domain, so that we can make sensible decisions on which process elements should be covered where and how. If our knowledge is poor or immature, we might initially experiment with a small core PML and many sub-PMLs. After collecting experiences, we are in a better position to decide – both strategically and technically – how the mutual sub-PML interfaces should be, which sub-PMLs could be reconciled, or which ones could be merged into the core PML. There is clear analogy with conceptual modelling of software systems: In newer approaches (see e.g. [vV92]), entity-relationship or object-oriented data models have been effectively combined with data flow diagrams or Petri-nets to describe the static and dynamic parts of the domain, respectively. Cf. also *federated databases*, trying to unify different data models, schemas and instances from possibly heterogeneous sub-databases.

However, sometimes we do not have a free design choice, since parts of the domain already are covered by existing or standardised languages and associated tools. The challenge is how the core PML and its PSEE shall or can *interoperate* with non-core languages and tools, e.g. for user interfaces, configuration management, or project management. This means that we should think strategically to prepare for co-existence of possibly inhomogeneous and partial process models. We may even stick to a small core PML to reduce labour and risk.

In other words, there are many factors to consider, when specifying and assessing the functionality of a PML to describe a given or desired process. In Figure 1 we group these factors along three main dimensions: meta-process phases (design, implementation etc.) process elements (activities, artifacts, ...), and user/tool interaction. In the "process element" dimension, there is a list of core and non-core process elements and their PML design approaches (**L1-L4**), which can be regarded as a summary of Section 2.3 and Section 2.4. In addition comes general PML design criteria, such as understandability and modularisation (Section 2.5). For all these dimensions, the EPOS PSEE and some other PSEEs are commented. However, the emphasis is on their positioning of the core PML vs. the sub-PMLs, not on comparing specific language paradigms.

The next Section 3 contains a more general discussion on PMLs in the perspective of interoperability, presenting a general PSEE architecture and an example of this.

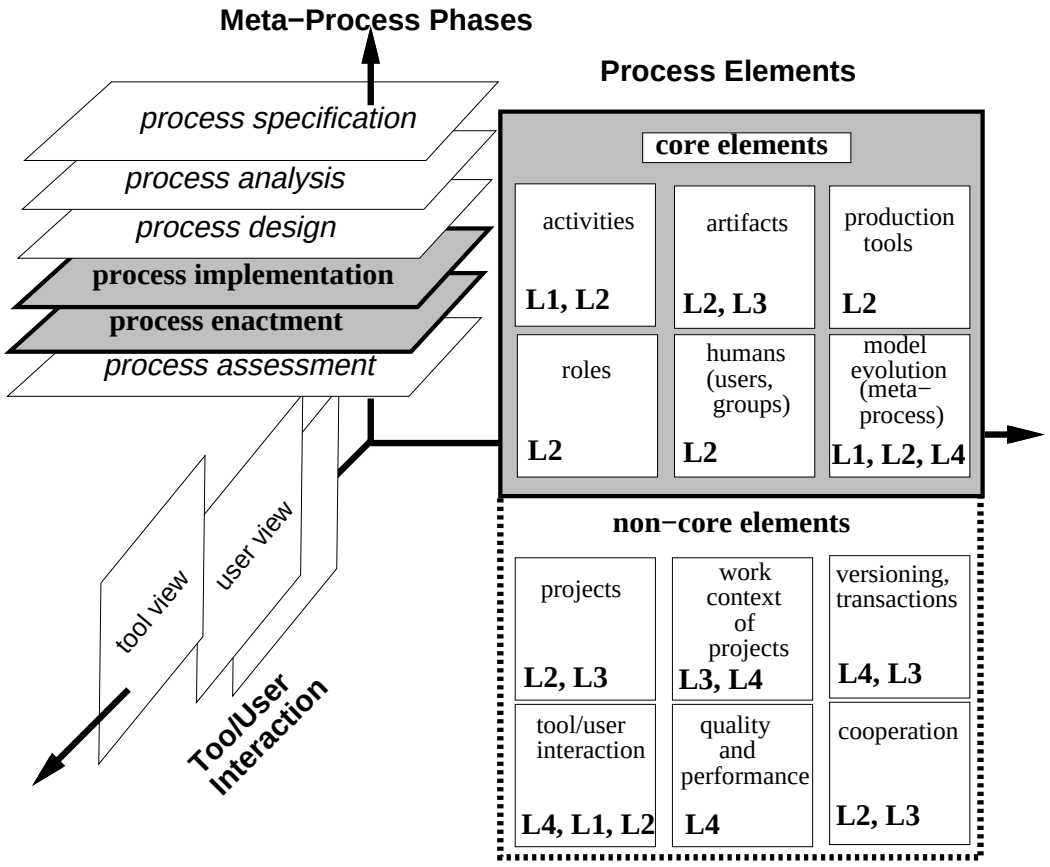


Figure 1: Three Dimensions of PML Design

## 2.2 PMLs and Meta-process Phases

The PML could be used to support the following meta-process phases:

1. **Process elicitation and requirement specifications**:
   Here, we must assist human understanding and negotiation of the perceived *as-is* process. We will need overall (conceptual) modelling, often stating business rules, coarse-grained work flow, and general work responsibilities. Intuitive and often graphical notations may be important if the audience is company decision makers. The PML of this phase will resemble languages used for Information Systems and Enterprise Modelling.

2. **Process Analysis**:
   Here, the PML must be sufficiently formal to be used for reasoning or simulation (dry runs). Changed needs from markets and new inputs from technology providers may enter in this phase. Decisions on changes to define a *to-be* process will typically take place here, so quality and performance matters must be dealt with here.

3. **Process Design**:
   Here, the PML must be able to express a more detailed process architecture and to incorporate more project-specific information, e.g. number of workpackages/subprojects, over-all planning (dependencies, timing), development technology (OO techniques), quality model etc.

4. **Process Implementation**:
   Here, the PML must allow specification of sufficient low-level details to achieve an enactable process model. This model must possibly be translated and otherwise prepared for execution. This may imply that parts of the process model is implemented and installed outside the PSEE, e.g. as tool configuration tables in a broadcast message server (BMS[1]) or as triggers/monitors in the production workspace.

5. **Process Enactment**:
   First, we start a Process Engine to achieve an enacting process model, residing in the PSEE repository. This Process Engine interacts with production tools and with human agents (wet runs). This interaction occurs, respectively, through a tool interface (e.g. a BMS) and a user interface (e.g. through an agenda).

6. **Process Assessment: Quality and Performance issues**:
   This covers anything from trivial follow-up of tool activations to collecting prepared measurements. All such data can be given as feedback to previous phases, either to guide the process or possibly to evolve the process model and its process. A Quality and Performance model must regulate all this, and the production tools must be properly instrumented for this purpose.

As shown, the different process lifecycle phases (the meta-process) exhibit different goals, contexts, and clients. It is doubtful that one PML can be used for all this, although we can use flow-based notations in most phases. Specialised and more high-level specification languages, with successive and not always automatic transformations towards more low-level enactable languages, is envisaged (the **L3** approach) – as for the normal software lifecycle. However, what is normally considered conceptual models in Information Engineering Systems are often being interpreted in PSEEs to support real enactment. For instance, specification languages like Petri-nets are used for such enactment.

In the following, we will only deal with PMLs for Process Implementation and Process Enactment. These are the only meta-process phases that we understand to some extent, but they also have the most detailed and heterogeneous information. Most PMLs and PSEEs, including EPOS, have emphasis on these phases.

[1] BMS is only used as an abbreviation, not as a potential product name.

## 2.3 PMLs and Process Elements to be Covered

A real-world software (production) process can be very large and have a variety of process elements. These must be modelled by a PML and supported by the associated PSEE. The PML should also describe the interface between the process support and the production process, covering tool activations, capturing of performance data, workspace set-up etc..

The next subsections will present a short-list of process elements (process subdomains), thus constituting a small **taxonomy** for such. These elements must be covered by a core PML, or by a collection of sub-PMLs. Such a division implies a core process model and many sub-models. As will be shown in section 3, such partial models could reside on separate model repositories or -servers.

In our opinion, the **core PML** must support the following **six** process elements: **activities**, **artifacts**, **roles**, **humans**, **production tools**, and **support for evolution (meta-process)**. The remaining process elements should be covered by sub-PMLs outside the core PML, but with clear interfaces both ways.

### 2.3.1 Activities (core)

Concurrent and cooperating activities are the heart of any process. They can be at almost any granularity level, and are usually associated to roles that can be filled by certain users and/or tools. Artifacts constitute the operands (inputs/outputs) of activities, so the core PML must contain a Data Manipulation Language (DML) to access such artifacts.

EPOS offers a predefined root task type (**L1**), that can be subtyped (**L2**) to express different model instances of tasks (in short: tasks). A task type is a normal object type but with special semantics attached to type-level attributes, such as PRE, POST, CODE, FORMALS and DECOMPOSITION (**L1**). External activities are modelled by cooperating and parallel transactions, containing internal tasks as co-routines. A task is runnable if its PRE-condition evaluates to `True`. The tasks stand in a chained and decomposed network, and is connected to artifacts, tools and users. A Planner tool helps in building such networks.

In SPADE [BFG93], an extended Petri-net formalism is used (**L1**) and with clustering of natural net units (**L2**). In Marvel [BK92], activities are described by a dual model: production rules resembling types (**L1**) and a separate network formalism (**L3**) to connect the rules and to attach the artifacts.

### 2.3.2 Artifacts (core)

The artifacts describe the product in question. In a reflective system, all process model fragments can be considered artifacts.

The product model will usually contain a basic data model, a product schema, and instances of the latter. At least product composition and dependencies must be described. We generally advocate an object-oriented paradigm (cf. CORBA [Obj92] and ODMG [Cat94]), in spite of lack of final standardisation.

On modelling of artifacts, we face a strategic choice. One alternative is to model a product as mere placeholders (like in Process Weaver, **L3**), in order to be independent of the actual product workspace. Another alternative is to model the product rather extensively as in EPOS, Adele and

Marvel. The latter allows some planning of project breakdown and suggestion of cooperation/ propagation protocols, all essential for process modelling. (Note, that Process Weaver is being coupled to Adele in the PERFECT ESPRIT project.)

EPOSDB has a structurally object-oriented data model, being extended by the SPELL PML to full object-orientation. SPELL is used to define a system-defined product schema (**L2**), supporting hierarchic families with own interfaces and bodies. Some utility tools utilise this product model and an external product description language is defined (a **L3** sub-PML), see below on workspace organisation in Section 2.3.8.

### 2.3.3 Roles (core)

A **role** defines responsibilities and rights for users that play that role.

In EPOS, a role is modelled as a separate model fragment, specifying certain access rights. Roles are then indicated by type-level attributes in task types (**L2**), and a task instance can only be connected to a user that can fill this role. No other semantics is utilised for roles, e.g. as in PWI.

### 2.3.4 Humans: Users and Groups (core)

A human **user** or process agent can fill a set of roles. He can also be a member of several **groups**, possibly nested, representing either project teams or line organisations.

In EPOS, users are modelled as `Person` instances, having a dynamic role-set (**L2**). This is checked against the role specification of the task, currently being performed by the user. No such groups are p.t. implemented.

### 2.3.5 Production Tools (core)

The tool model must specify how such tools can be accessed and controlled. We must distinguish between batch and interactive tools, and be able to handle call-backs from both.

In EPOS, a production tool is modelled as a low-level task type, specifying I/O, command line formats, options etc. (**L2**). A tool is started by sending a detailed message from the task's CODE part. The response is returned through an extra task input, and causes the internal task to be reactivated.

SPADE offers more flexibility to describe tool behaviour as a cluster of Petri-net transition nodes (still **L2**).

### 2.3.6 Support for Evolution or Meta-process (core)

Due to the human-oriented nature of the software process, we have an inherent cause for evolution during process enactment. This means that most previous lifecycle phases must be repeatable "on-the-fly". Thus the core PML must offer support for evolution of at least the process model, both technically (e.g. by reflection or interpretation) and conceptually (by a defined meta-model). Evolving an *enacting* model is known as "pulling the rug" from telecommunication switches, relying on dynamic linking and special hardware.

The available PML mechanisms to support model evolution differ, e.g. by predefined facilities (delayed subtask expansion in MELMAC [GJ92], **L1**), by a fully reflective PML (as in EPOS, SPADE, and PWI – **L2**), by another language (as in Marvel, **L4**), or ad-hoc (Process Weaver, no support). Note, that parts of the process model may exist in many different (translated) forms in a PSEE and its environment, cf. Section 2.3.10. Thus, to have reflection or interpretation in the core PML, as implemented by some PSEEs, is grossly insufficient.

However, none of the PSEEs deal explicitly with evolution of the real-world production process, also being an enacting and concurrent body.

### 2.3.7 Projects (non-core)

A project contains a variety of domain-specific information. A project model thus might include: a workplan with sub-activities/-projects, responsible for activities, overall goals and inputs/outputs, available resources, time estimates, work records (time sheets, logs etc.), quality model (Section 2.3.9), cooperation patterns between projects, and connection to workspace transactions and versioning (Section 2.3.10). Such project information is revised almost daily (i.e. it is highly "versioned"!), both to record ongoing work and to make adjustments based on this.

Evidently, some of this information (**L3**) overlaps with or is translatable down to the core process model (**L2**).

In EPOS, a project model fragment is of the `project` task subtype (**L2**). It is connected 1:1 to a PSEE repository transaction (**L3**).

### 2.3.8 Work Context of a Project (non-core)

This includes a production workspace and the available production tools.

A **production workspace** (e.g. files or a CASE-tool repository) is the external representation of a configuration, which again is a requested part of the total, versioned product. A mapping and check-out/check-in between the internal PSEE repository and the external workspace may be needed. The **production tools** have possibly been instrumented with interfaces to the process tools.

An EPOS production workspace is a set of files checked out from the EPOSDB repository through a Workspace Manager, which has a special check-out tool and an EPIT production description language (**L3**). The configuration specification and the mapping from the repository to the workspace is described in special languages (**L3** and **L4**), being a part of the previous project model (Section 2.3.7). In case of overlapping workspaces, the Cooperation Manager must be activated (Section 2.3.11). Production tools (e.g. compilers, editors) can then work on these files, guided by the PSEE. Before transaction commit, the modified files must be brought back through a similar check-in tool.

### 2.3.9 Product Quality and Process Performance (non-core)

A **product quality model** includes operational goals of product quality and associated metrics, e.g. review and test status [IS94]. The **process performance model** for process quality expresses compliance to the stated process model, e.g. wrt. deadlines, budget, and user roles.

EPOS has none of this now, but this is being worked upon in two PhD theses.

### 2.3.10 Versioning and Transactions (non-core)

Versioning at least of the production workspace is needed, and likewise with some support for long transactions. Whether such support should be extended to also encompass the PSEE repository is an open issue.

The chosen **versioning model** should allow transparent versioning during a transaction. It should be orthogonal to the process model, so that the same versioning can apply uniformly to all process model fragments. Note, that there are parts of the process support, that cannot easily be subjected to formal versioning and transactions. We can mention configuration tables in BMSes, "shell-envelopes" around production tools, or extra triggers in production workspaces.

EPOS rely on Change-Oriented Versioning [L$^+$89] [Mun93], giving uniform versioning. Process modelling and versioning work well together (**L4**). On the one hand, a particular version of a partial product can be checked out into the workspace before the process starts, using EPOSDB facilities and the added check-out tool (Section 2.3.8). On the other hand, the EPOS process model is itself a first-class artifact, and can be stored, versioned and reused by the same versioning model.

In Adele2 [BEM93], the product model instances (product descriptions) are supported by one versioning model. For the product schema and for the activity/tool model, other facilities are used to achieve evolution, e.g. special delegation paths and roles (meaning versioned types).

The **transaction model** should be nested and allow pre-commit cooperation. Preferably, the PSEE repository should contain an instrumentable transaction model, with "hooks" into the process model in case of certain events. EPOS has extensive support for cooperating transactions, where high-level cooperation protocols (**L3**) are translated down to e.g. operational triggers and task networks. Similar has been demonstrated by Adele. EPOS also provides some planning tools for transaction (or project) breakdown, scheduling, and cooperation [CHL94].

### 2.3.11 Cooperation (non-core)

We have two basic modes of cooperation: *sequential*, e.g. by normal work or review chains, or *parallel*, e.g. upon workspace overlap.

For sequential cooperation, the normal EPOS task networks are used, also across transactions. For parallel cooperation, EPOS has a Cooperation Manager that helps in setting up high-level cooperation policies (**L3**) between projects/transactions. These are then translated into more low-level support (triggers, task networks, inter-transaction propagation – i.e. **L2**), using the extended transaction concept of EPOSDB (Section 2.3.10).

Using roles to express interaction diagrams has been investigated by e.g. PWI, but not by EPOS.

## 2.4 PMLs and Tool/User Interaction Paradigm (non-core)

This deals partly with how and to what degree the process support intervenes with the user's normal way of working, called the *tool view* (or PSEE tool coupling). This is only relevant in the last lifecycle phases.

It also deals with how parts of the process model should appear through a *user view*, both conceptually and graphically. This applies in all lifecycle phases, but especially in the last ones.

### 2.4.1 Tool View (non-core)

The tool view deals with how much the process support "perturbates" the production process. We can identify at least two different tool views or work modes [FG94]:

1. In most existing PSEEs, the user has a *task-oriented view* of the process: The user is directed by the PSEE. That is, most tool activations are strictly controlled and invoked through a central process support interface. That means that all relevant production tools must be enveloped.

2. More preferably, the user needs a *goal-oriented view*: After setting a goal, he can move freely in the process space to achieve the goal. The process support will listen to events in the production process, and give guidance based on this. Again, the production tools must be (invisibly) enveloped, or all relevant accesses to the production workspace must be trapped to inform the PSEE.

The tool view often gets embedded in the PSEE, both conceptually and technically, although this is hardly ever explicitly expressed in the PML (thus **L1**). Is might have been possible for the user to select between the two above alternatives (assistance vs. control), and even to do this dynamically (!).

EPOS allows either manual or automatic activities to be expressed in SPELL. The former are approved through an agenda, that contains runnable activities with `True` PRE-conditions (partly **L2**).

The Marvel Process Engine only approves user actions (**L1**), but cannot itself initiate actions behind the user's back, not even recompilations.

### 2.4.2 User View (non-core)

The user view or user interface is a crucial, but often neglected part of many PSEEs, including EPOS. This fact is amplified by the delicate balance between "control" and "assistance", cf. the previous subsection. However, the problem of displaying complex and heterogeneous information to users with different levels of competence and goals, are shared by most computerised systems [Mye89].

The general paradigm of user interfaces is that we should split *how is works* (internal model, e.g. in C or Prolog) from *how to do it* (external view) – i.e. **L3** or even **L4**. Some aspects to consider are: uniformity of presentation and interaction, easy comprehension of presented information, and flexible choice of presentation formats (filters, viewers). For instance, the external view could be a task network, while the internal activity model is rule-based – or inversely! Cf. also the ECMA roaster model [ECM91], where the user interface is made separate. This separation has a tactic consequence: first we should define by fast prototyping the interface on the user's premises and based on familiar concepts according to his role. Then we may consider to implement this interface by some suitable technology.

Much research has been devoted to expressing multiple and possibly incompatible model views to the user. The views can be conceptual (using ViewPoints [NKF93]) and/or graphical (using Smalltalk's Model-View-Controller [GR83]). For the latter, [Fol83] recommends showing simply the entities and their relations and operations.

EPOS provides the user with an activity network and an agenda. The network is a simple print-out

of the internal data of the process engine. A viewing filter between the engine and the user is needed, providing alternate views according to user roles.

### 2.5 General PML Demands (core)

**Analysability**: A PML, as most modelling languages, should be sufficiently formal to allow precise modelling, analysis and simulation. Most PMLs have this property, including the EPOS SPELL.

**Understandability**: A PML should be user-oriented and easy to comprehend. This may assume informality to some extent, or separate presentation languages (**L3**). However, most PMLs are too low-level and technical, including the EPOS SPELL.

**Modularisation**: It should offer modularisation features to support grouping and protection, and sub-model constructs to facilitate reuse on a rather high level, e.g. as sub-schemas (**L2**). This point is further elaborated in the next section on interoperability (**L3** and **L4**). EPOS allows models to be organised in task hierarchies, e.g. according to projects.

### 2.6 A Short Summary

A short summary of the core and non-core process elements and their sub-PML design approaches discussed in this Section can be found in Figure 1, along the "Process Elements" dimension. Note, that many PSEEs do not conform to the recommended core PML design, as **L1** and **L2** constructs are used to express non-core process elements.

## 3 A General Perspective on PML Design: Interoperability

As mentioned, we may need a core PML and a set of sub-PMLs to cover the variety of process elements, or to interface to existing and domain-specific models, their modelling tools and model repositories. In most PSEEs, all the (sub)models are stored in a common repository, so it is worth to consider segmentation also of this repository.

Figure 2 shows a general PSEE architecture with a segmented repository, or rather a collection of **model servers**. A relationship model server could be added to store inter-model references. The segmentation could be conceptual with all sub-models still residing in a single repository, or physical with some sub-models in separate repositories. This means that the associated PSEE tools become tools in their own right, not just local procedures in a common process tool. Note that the ECMA CASE environment architecture is applied throughout, with separate database-, tool- and user interfaces.

In our opinion, most of these sub-models should, at least representationally, be defined by a *(structurally) object-oriented* data model. The models should be accessible to the PSEE tools through a standard object-oriented database interface, as defined by CORBA or ODMG.

This section considers interoperability between the different sub-models and the PSEE tools, and between these and the production workspace/tools. The emphasis is on the implication to PML design.

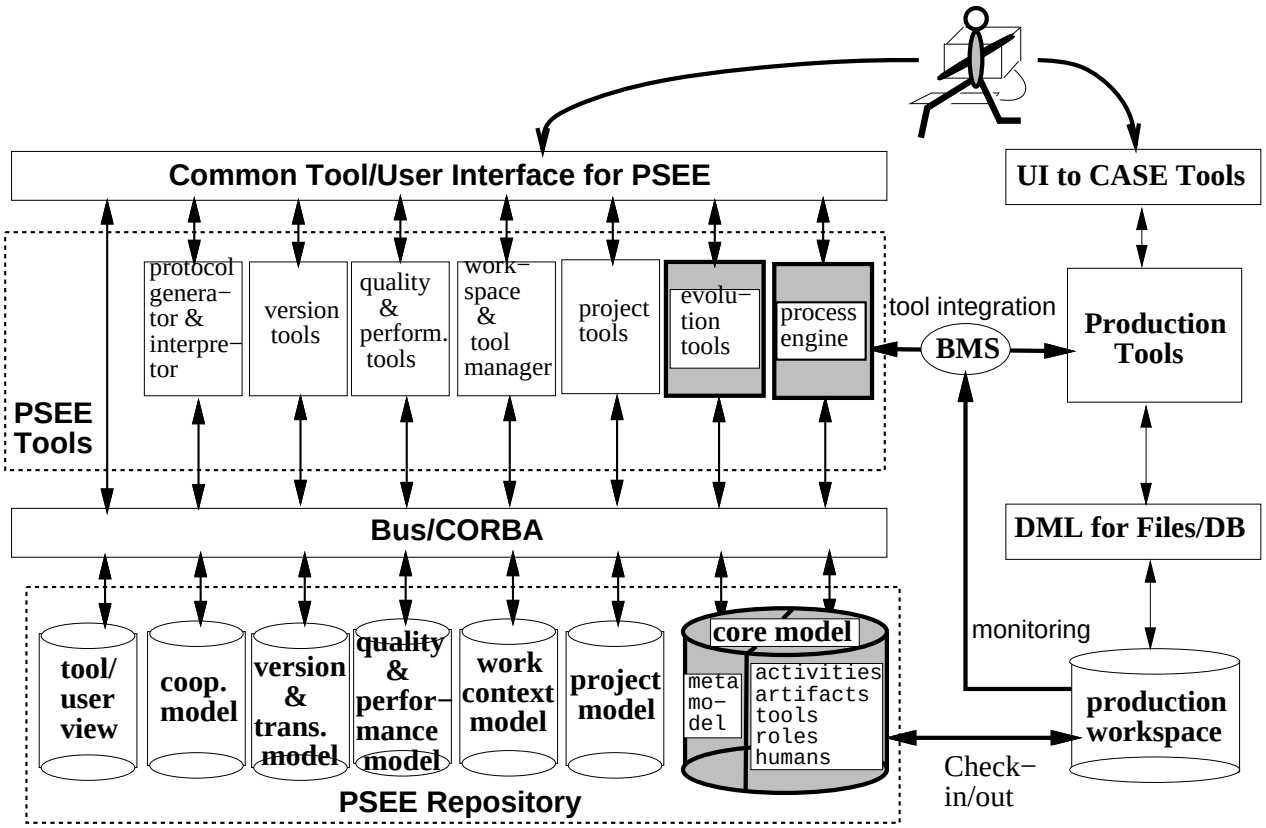


Figure 2: PSEE Architecture with Segmented Repository.

## 3.1 Interoperability Between the Core Model and the Other Sub-models

### 3.1.1 Core vs. Project Model

As mentioned, interoperability between the core model and the project sub-model can be achieved by the **L3** strategy of PML design. This is done in EPOS and Process WEAVER.

However, often an existing project management tool must be incorporated into the PSEE, and this tool is often instrumented to the client organisation. In this case, EPOS must give up most of its project sub-model, and try to integrate (still **L3** level) with the former. In Section 3.2 we will present a real-world example of this.

### 3.1.2 Core vs. Work Context Model

The PSEE repository is almost always a software engineering database (DB), preferably object-oriented. We can identify three possible combinations of the PSEE repository and the production workspace:

1. **PSEE Repository vs. File System**:
   As mentioned before, one alternative is to do check-out and check-in of files from the PSEE repository as part of a transaction, and to ensure some level of mutual consistency during this transaction. All this require specifications in and translation from a **L3** PML.

   Another alternative is to rely on a *virtual* file system, where all file accesses are trapped

and mapped directly onto the PSEE directory, cf. PCTE [BGMT88] or ClearCase [LL93]. Transparent versioning is also easy to arrange with such a solution.

2. **PSEE Repository vs. own Sub-Repositories**:
   The "cleanest" combination is that the production workspace is a sub-database of the core PSEE repository. However, CORBA and ODMG technologies are immature, hence few production tools have adapted to these so far.

   Of PSEEs, Adele and EPOS have made the workspace *conceptually* a sub-database of a single repository. Most DBMS commands can then be carried out in this workspace, while external tools can continue to work on checked-out external files.

3. **PSEE Repository vs. separate DB(s)**:
   This resembles the first alternative in Section 3.1.2, but we normal check-out and check-in against the PSEE repository is not realistic. We must therefore create a "shadow" product model in the PSEE repository, as for the first alternative and as in the example in Section 3.2. Generally, this brings up the question of federated databases, but this is not dealt with for obvious reasons.

In all these three combinations, an object-oriented paradigm seems appropriate for the PSEE-internal artifact model, i.e. **L3**.

### 3.1.3 Core vs. Quality/Performance Model

The product quality model is largely independent of the core model, while the performance model is more tightly linked.

### 3.1.4 Core vs. Versioning/Transaction Model

Our EPOS experience advises uniform and transparent versioning to the entire PSEE repository in a transaction context. However, most standard DBMSes or file systems do not offer such. Thus, the next-best solution is special versioning of certain parts of the repository or only for a file-based production workspace.

Versioning/transaction description languages (**L3/L4**) and tools are therefore pervasive technologies. They are linked to the core process model and also to the project and cooperation model.

### 3.1.5 Core vs. Cooperation Model

Cooperation is rather enhanced by interoperability, but the basic cooperation model should not be affected. An increasing number of groupware or workflow systems are also becoming available, with own languages (**L3/l4**) and paradigms. To make these accessible in the context of a PSEE will be a challenge.

### 3.1.6 Core vs. Tool View Model

The prevailing tool interaction technology is to use point-to-point (e.g. CORBA) or broadcast message servers (e.g. FIELD) to let tools cooperate in a flexible way. By proper enveloping, tools need not be aware of that they are "coordinated", nor do the PSEE's tool model or Process Engine

be bothered by low-level details here. This means that tool descriptions are partly represented in the PSEE tool model and partly in BMS-internal configuration tables (**L3**).

Interactive tools like Emacs are difficult to handle, partly because they can do almost anything. It is, however, possible to trap and instrument their file accesses, or to rely on more fine-grained tool-tool interfaces.

#### 3.1.7 Core model vs. User View Model

As mentioned, the User View or User Interface (UI) model is a critical part for a PSEE. The underlying linguistic paradigm of a PML is judged not crucial for a good UI (**L4**).

There are large savings by using existing technologies for user interfaces, as exemplified by X and Motif, UIMSes, graphical browsers and editors, and report generators. Thus, the interface between the User Interface and the PSEE tools should be standardised (**L3**?). Then we can more easily use standardised user interface technologies, and plug new process tools to the PSEE user interface.

### 3.2 An Example of a Segmented PSEE

The SYSDECO software house in Oslo has developed and is selling a 4G-tool called Systemator, running on Unix workstations. Using this 4G-tool, applications can be written in a special Sysdul language. From Sysdul "programs", Systemator can generate complete user applications against a given, commercial database.

Sysdeco is also developing customer applications in Systemator on behalf of customers, say in Project X. They use no computerised project management system to support such projects. A MicroSoft/Project tool on PCs is in use only by Project Managers, first to make a coarse plan – with activities, schedules and personnel – and then to manually record ongoing activity status (time sheets etc.). Most project data are recorded on paper. A quality model is defined, but again followed up manually.

Below, we will recapitulate the projected design of a PSEE demonstrator to offer more automatic support for such development projects. The work is done at NTH by a student project group, lead by Geir Magne Høydalsvik [Hø94]. For this demonstrator, Sysdeco insists on using only commercially available tools and databases for this, including their own Systemator tool. Thus, there is initially no room for EPOS.

Figure 3 shows the projected and segmented PSEE for this project support. The part drawn in slender lines represents the PSEE, while the part drawn in the bold or dotted lines represents a possible generalisation using an extended EPOS. Thus, the projected PSEE has the following components, where the actual PML being used is indicated as $L_{...}$:

1. **Systemator 4G-tool**, utilising $L_{SYSDUL}$ and generating applications against a **product database** on top of the Ingres DBMS. This is the same as before, but triggers must be inserted in the database (expressed in $L_{INGRES\_DML}$) to send messages to Process WEAVER upon certain events.

2. **A MicroSoft/Project** tool, initially used by a Project Manager to produce a coarse project plan, resembling a Gannt diagram. This initial plan is printed out in some intermediate, legible format ($L_{MS/P}$). This plan is then manually extended into a full project model (in

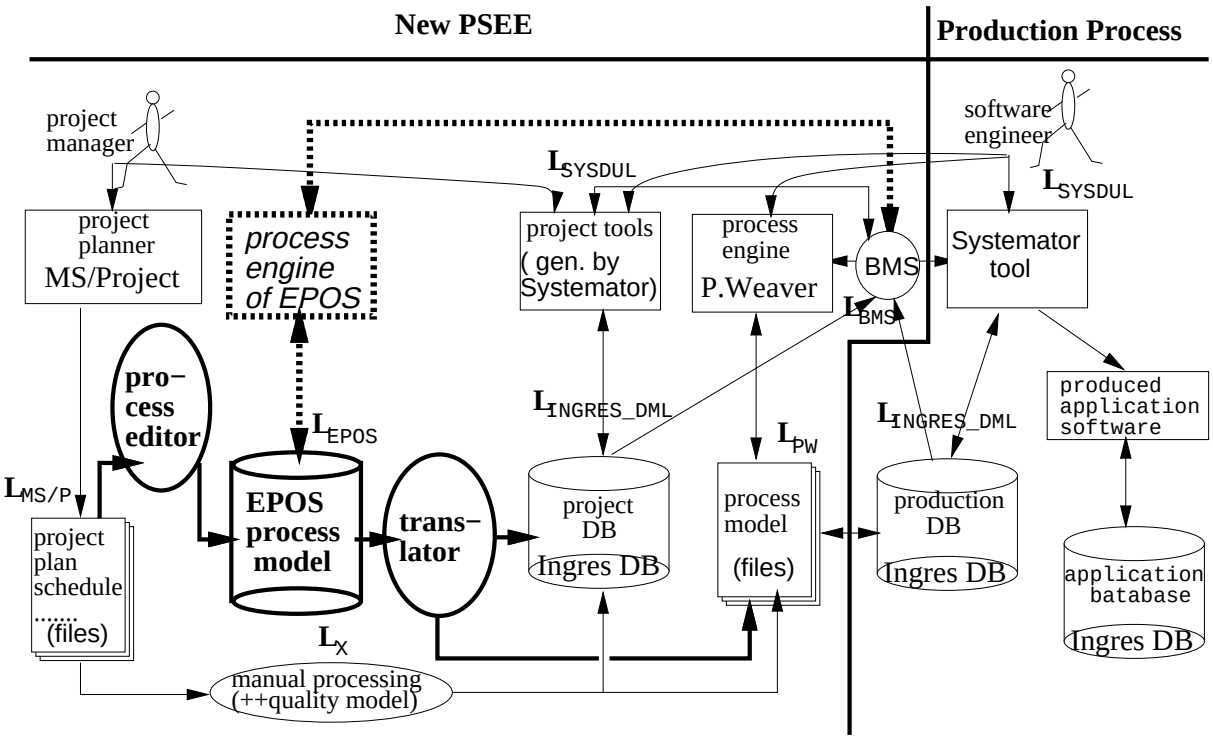


Figure 3: Design of a PSEE for Project Support

"$L_X$"), including an enhanced quality model. Probably, "$L_X$" will not be an entirely formal language.

3. **Project tools**, using a project database also implemented by Ingres. These tools will initially be generated by the same Systematator tool (and expressed in $L_{SYSDUL}$). The project database must similarly be instrumented by triggers (in $L_{INGRES\_DML}$).

   These tools will first be used to install the full project/quality model. Later, the developers themselves will be filling in quality and performance data during normal development. Likewise, product data are being fetched from the product database, possibly through triggers.

4. **Process WEAVER**, being the Process Engine and using files as its process model repository (in $L_{PW}$). Process WEAVER will communicate both with the project tools and their project database, and with Systematator and its product database. Its process model will be manually made, based on the full project/quality model.

5. A **Broadcast Message Server** (configured by $L_{BMS}$), to link Process WEAVER with the other tools and databases.

6. Various graphical presentation languages ($L_{UI}$), not elaborated further.

All of these languages are classified as **L3** languages wrt. a core PML, perhaps with exception of $L_{PW}$ being an enactable PML in its own right.

Our first observation is that this PSEE design fits very well with the general PSEE architecture from Figure 2. Let us then consider how more advanced process modelling techniques such as EPOS can be applied to improve (or enrich) the projected PSEE.

We can rather build a complete process model in a possibly extended SPELL ($L_{EPOS}$), based on the extended process/quality model. This combined and more high-level process model can then be translated into the above sub-models in their actual sub-PMLs, thus replacing the current manual processing. Indeed, we need a powerful **PSEE meta-tool** or CASE tool for process modelling. This should be able to build, translate, install and maintain a set of distributed and heterogeneous process models, used by partly commercial and standardised process tools! Note a possible bootstrap (meta-process), as the goal of the application project could be to make a new version of the project tools!

We can also use our EPOS process engine (in dotted lines of the figure) to instrument the production process. Of course, in this case we have to achieve interoperability with the existing project model.

# 4 Conclusion

Until recently, much research in process modelling has been focussed on different linguistic paradigms for the core PML, in order to find *the* correct one. This paper addresses the "one or many PMLs" question from a broader point of view.

To recapitulate: there is a big variety of process phases and elements to be covered, although we focus only on the Imnplementation and Enactment phases. We must also interface towards actual production tools/workspaces. Different user roles have different needs, also wrt. work modes and user interfaces.

There are many technical arguments behind choosing *one core PML* (**L1/L2** approach) and *a set of sub-PMLs* (**L3/L4** approach). However, the decisive factor in choosing the "federated" or "interoperable" approach, is that we *have to* adapt to a myriad of relevant through alien languages, tools, and databases. All of these must somehow be incorporated into or interfaced against the PSEE. That is, the PSEE developers simply do not control the PML design space. Thus, interoperability against standard or existing subsystems is an *absolute must*, specially since process support should be an add-on to existing computerised production tools, not a hindrance.

We even claim that the choice of the underlying linguistic paradigm for such a core PML (which must cover the short-list in Section 2.3) is not so important. What really counts is:

- **Standardisation**:
  Use of standard support technologies: Unix/MS-DOS, C++, CORBA or OBMG, X/Motif, new workflow systems, etc. etc.. Reuse is a keyword here.
- **Interoperability**:
  Making PSEE components interact smoothly with other process and production models and tools. Modularisation, open systems and above standardisation are keywords here.
- **Tool view (PSEE coupling)**:
  The PSEE should "perturbate" a software production environment in a minimal way. The main goal of the PSEE should be to give flexible enactment support at the appropriate level, e.g., control, automation, guidance, reasoning, explaining etc..
- **User view (user interface)**:
  The process agent should be presented a comprehensible view of his current work context, with proper connections to the co-workers' activities. Been given this, the agent can (more) intelligently execute and relate to his own role in the overall process.

- **Easy user-level evolution of the process model**:
  Again, the goal is to provide an understandable view of the model, so that this can be changed by the process agents themselves, if and when needed.

**Acknowledgements**

Thanks go to colleagues in the PROMOTER project, and to the local teams behind EPOS.

# References

[ABGM92] P. Armenise, S. Bandinelli, C. Ghezzi, and A. Morzenti. Software Process Representation Languages: Survey and Assessment. In *Proc. 4th IEEE International Conference on Software Engineering and Knowledge Engineering, Capri, Italy, June 17-19. 31 pages*, June 1992.

[ACF94] Vincenzo Ambriola, Reidar Conradi, and Alfonso Fuggetta. Experiences and Issues in Building and Using Process-centered Software Engineering Evironments, September 1994. Internal draft paper, Univ. Pisa / NTH, Trondheim / Politecnico di Milano, 26 p.

[BBFL94] Sergio Bandinelli, Marco Braga, Alfonso Fuggetta, and Luigi Lavazza. The Architecture of the SPADE Process-Centered SEE. In *[War94]*, pages 15–30, 1994.

[BEM93] N. Belkhatir, Jacky Estublier, and Walcelio Melo. Software Process Model and Work Space Control in the Adele System. In *[Ost93]*, pages 2–11, 1993.

[BFG93] Sergio Bandinelli, Alfonso Fuggetta, and Carlo Ghezzi. Software Process Model Evolution in the SPADE Environment. *IEEE Trans. on Software Engineering*, pages 1128–1144, December 1993. (special issue on Process Model Evolution).

[BGMT88] Gerard Boudier, Ferdinando Gallo, Regis Minot, and Ian Thomas. An Overview of PCTE and PCTE+. In *Proc. of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, Boston, Massachusetts, November 28–30*, pages 248–257, 1988.

[BK92] Naser S. Barghouti and Gail E. Kaiser. Scaling Up Rule-Based Development Environments. *International Journal on Software Engineering and Knowledge Engineering, World Scientific*, 2(1):59–78, March 1992.

[C+94] Reidar Conradi et al. EPOS: Object-Oriented and Cooperative Process Modelling. In Anthony Finkelstein, Jeff Kramer, and Bashar A. Nuseibeh, editors, *Software Process Modelling and Technology (PROMOTER book)*, pages 33–70. Research Studies Press/John Wiley & Sons, 1994.

[Cat94] Rick G. G. Catell. *Object Data Management: Object-Oriented and Extended Relational Database Systems*. Addison-Wesley, 1994.

[CFF94] Reidar Conradi, Christer Fernström, and Alfonso Fuggetta. Concepts for Evolving Software Processes. In Anthony Finkelstein, Jeff Kramer, and Bashar A. Nuseibeh, editors, *Software Process Modelling and Technology (PROMOTER book)*, pages 9–32. Research Studies Press/ John Wiley & Sons, 1994. Also as EPOS TR 187, NTH, 9 Nov. 1992, 26 p., Trondheim.

[CFFS92] Reidar Conradi, Christer Fernström, Alfonso Fuggetta, and Robert Snowdon. Towards a Reference Framework for Process Concepts. In *[Der92]*, pages 3–17, 1992.

[CHL94] Reidar Conradi, Marianne Hagaseth, and Chunnian Liu. Planning Support for Cooperating Transactions in EPOS. In *Proc. CAISE'94, Utrecht*, pages 2–13, June 1994.

[CLJ91] Reidar Conradi, Chunnian Liu, and M. Letizia Jaccheri. Process Modeling Paradigms. In *Proc. 7th International Software Process Workshop – ISPW'7, Yountville (Napa Valley), CA, USA, 16–18 Oct. 1991, IEEE–CS Press*, pages 51–53, 1991.

[Der92] Jean-Claude Derniame, editor. *Proc. Second European Workshop on Software Process Technology (EWSPT'92), Trondheim, Norway. 253 p.* Springer Verlag LNCS 635, September 1992.

[ECM91] ECMA. A Reference Model for Frameworks of Computer Assisted Software Engineering Environments. Technical report, European Computer Manufactoring Association, 1991. ECMA/TC33 Technical Report, Nov. 1991, Draft Version 1.5.

[Fer93] Christer Fernström. Process WEAVER: Adding Process Support to UNIX. In *[Ost93]*, pages 12–26, 1993.

[FG94] Alfonso Fuggetta and Carlo Ghezzi. State of the Art and Open Issues in Process-Centered Software Engineering Environments. *Journal of Systems and Software*, 26(1):53–60, July 1994.

[FH93] Peter H. Feiler and Watts S. Humphrey. Software Process Development and Enactment: Concepts and Definitions. In *[Ost93]*, pages 28–40, 1993.

[Fol83] J. D. Foley. Managing the Design of User Computer Interface. *Computer Graphics World*, pages 47–56, December 1983.

[GJ92] Volker Gruhn and Rüdiger Jegelka. An Evaluation of FUNSOFT Nets. In *[Der92]*, pages 196–214, 1992.

[GR83] Adele Goldberg and Dave Robson. *Smalltalk-80: The Language and its Implementation*. Addison-Wesley, 1983. 714 pp.

[Hø94] Geir Magne Høydalsvik. Programmering Prosjektarbeid, Forslag til Prosjektoppgave. (In Norwegian, Working note for PhD thesis), August 1994.

[IS94] IEEE-Software. Special Issue on Measurement-based Process Improvement. *IEEE-Software*, July 1994.

[JC93] M. Letizia Jaccheri and Reidar Conradi. Techniques for Process Model Evolution in EPOS. *IEEE Trans. on Software Engineering*, pages 1145–1156, December 1993. (special issue on Process Model Evolution).

[L+89] Anund Lie et al. Change Oriented Versioning in a Software Engineering Database. In *Walter F. Tichy (Ed.): Proc. 2nd International Workshop on Software Configuration Management, Princeton, USA, 25-27 Oct. 1989, 178 p. In ACM SIGSOFT Software Engineering Notes, 14 (7)*, pages 56–65, November 1989.

[LL93] Paul H. Levine and David Leblang. Software Configuration Management: Why is it needed and what should it do? In *Proc. of 4th International Workshop on Software Configuration Management (SCM-4), Baltimore*, pages 174–179, May 1993.

[Lon93] Jacques Lonchamp. A Structured Conceptual and Terminological Framework for Software Process Engineering. In *[Ost93]*, pages 41–53, 1993.

[LSS94] Odd Ivar Lindland, Guttorm Sindre, and Arne Sølvberg. Understanding Quality in Conceptual Modelling. *IEEE Software*, pages 42–49, March 1994.

[Mun93] Bjørn P. Munch. *Versioning in a Software Engineering Database — the Change Oriented Way*. PhD thesis, DCST, NTH, Trondheim, Norway, August 1993. 265 p. (PhD thesis NTH 1993:78).

[Mye89] Brad A. Myers. User-Interface Tools: Introduction and Survey. *IEEE Software*, 6(1):15–23, January 1989.

[NKF93] Bashar Nuseibeh, Jeff Kramer, and Anthony Finkelstein. Expressing the Relationship between Multiple Views in Requirements Specification. In *Proc. 15th IEEE Int. Conf. on Software Engineering (ICSE)*, May 1993.

[Obj92] Object Management Group, 492 Old Connecticut Path, Framingham, MA 01701, USA. *OMG CORBA Common Object Request Broker Architecture – Specification*, January 1992.

[Ost93] Leon Osterweil, editor. *Proc. 2nd Int'l Conference on Software Process (ICSP'2), Berlin. 170 p*. IEEE-CS Press, March 1993.

[Rob94] Ian Robertson. An Implementation of the ISPW-6 Process Example. In *[War94]*, pages 187–206, 1994.

[SC92] H. G. Sol and R. L. Crosslin, editors. *Dynamic Modelling of Information Systems 2*. North-Holland, 1992.

[vV92] Kees M. van Hee and P. A. C. Verkoulen. Data, Process and Behaviour Modelling in an Integrated Specification Framework. In *[SC92]*, pages 191–218, 1992.

[War94] Brian Warboys, editor. *Proc. Third European Workshop on Software Process Technology (EWSPT'94), Villard-de-Lans, France. 274 p.* Springer Verlag LNCS 772, February 1994.

epos/papers/pml-ewspt95.tex

liu/PMseg/liu.tex