

PSEE Architecture:
EPOS process models and tools

Reidar Conradi et al.
Norwegian Institute of Technology, N-7034 Trondheim,
Norway

Phone: +47 73 593444, Fax: +47 73 594466, Email: conradi@idt.unit.no

February 16, 1996

2nd Workshop on Process-centered Software Engineering
Environment (PSEE) Architectures

Univ. California, Irvine

20–23 Feb., 1996

Contents

1	Summary	6
2	Introduction	7
3	EPOS context, history and rationale	8
3.1	EPOS context and history	8
3.2	Rationale and design criteria EPOS PM	9
4	EPOS by Examples	10
4.1	EPOS Architectural Layers	10
4.2	EPOSDB and PM application	11
4.3	EPOS distributed architecture	12
4.4	EPOS process model architecture	13
4.5	ISPW7 reference example in EPOS	14

5	EPOS Model Architecture	15
5.1	The EPOS PML: SPELL	17
5.2	BA.ACT: Activity or Task Submodel	20
5.3	BA.PROD: Product submodel	22
5.4	BA.HUM: Human/role submodel	23
5.5	PR.ORG: Organization/project submodel	24
5.6	US.TOOL: Tool submodel	26
5.7	US.WS: Workspace submodel	28
5.8	US.INTE: Tool Integration submodel	28
5.9	US.UI: User Interface submodel	28
5.10	VA.EVOL: Meta-process submodel	29
5.11	VERS: The versioning submodel	31
5.12	TRANS: The transaction submodel	37
5.13	General Model Organization and Evolution	42

6	EPOS Tool Architecture	44
6.1	General architecture and examples	44
6.2	EPOSDB	46
6.3	SPELL Interpreter: Low-level Process Engine .	47
6.4	Execution Manager: Low-level Process Engine	48
6.5	Planner	51
6.6	Schema Manager	52
6.7	Project Manager	53
6.8	Workspace Manager	57
6.9	Cooperation Manager	59
6.10	PM Assistant (new)	60
7	Experiences with EPOS model/tool architecture	61
7.1	Experiences with EPOS process models	61

7.2 Experiences with EPOS tools	62
8 Conclusion	63

1 Summary

- Introduction
- EPOS context, history, rationale
- EPOS by examples
- EPOS process model architecture
- EPOS process tool architecture
- EPOS experiences
- Conclusion

2 Introduction

- Use EPOS to illustrate requirements and choices for model and tool architectures in PSEEs.
- EPOS emphasis: uniformity, OO, planning, reflectivity and meta-process, cooperating transactions, CM+PM.
- EPOS status wrt. implementation and use.
- Lessons learned.
- Conclusion

3 EPOS context, history and rationale

3.1 EPOS context and history

- Open-ended, kernel **PSEE**.
- **CM: Manage evolving software products.**
Original **change-oriented versioning (COV)**,
client-server **EPOSDB**, with structurally OO DDL.
On Unix w/RPC and C-ISAM, in C, with Prolog client DML.
- **PM: Manage associated processes, process modeling.**
Active PM extensions upon *passive* EPOSDB.
SPELL: Full OO, type-level info, meta-types.
Schema Manager, PM Assistant (new),
Process Engine and Planner,
Project Manager, Cooperation Manager and
Workspace Manager.
In SWI-Prolog using XPCE graphics and a BMS.
Common SPELL for products and activities, and meta-PM.
- Expert system for Program and (“Og”) System development.
Contract ED0224.18457, NTNF R&D Council (1986-91),
part of national IT-program.
- At NTH and partly DELAB in Trondheim (1986-95).

3.2 Rationale and design criteria EPOS PM

- **Coverage:** Entire software life-cycle, open-ended.
Process support by model definition, enaction, evolution;
both “soft” guidance and “hard” automation/control.
Automatic and manual activities, and hybrids.
Multi-actor, distributed environment.
Coordination / communication: intra/inter-transactions.
- **Typed network** for activity model;
not only implicit rules and triggers.
Procedure (+triggers): fine-grained operations.
Activity: coarse-grained operations.
- **Static** reasoning and planning (FORMALS, DECOMP.),
Dynamic execution (PRE, CODE) for “active” DB.
- **Model structuring:**
 - general instances with OO types/meta-types,
 - type inheritance,
 - instance-level composition of product/activity/project,
 - type-level composition by network templates,
 - sub-projects with versioning.
- **Customization and evolution:**
Reflection by meta-types, OO dynamic binding / polymorphism, universal versioning of projects (PM+CM), overwrite, ...
- **Prototyping** environment, Prolog bias. Unix and X.

4 EPOS by Examples

4.1 EPOS Architectural Layers

See also [COWL91] [C⁺94] [MCL⁺95]:

1. **Client-server, uniformly versioned EPOSDB.**
Structurally OO DDL, Prolog DML, TypeDescriptors.
Nested, coop. transactions; version = *visible* sub-DB.
2. **SPELL, behaviourally OO PML extension in Prolog.**
Full OO: DDL+DML, PROCs/triggers, type-level attrs, meta-types.
PM Manager with SPELL Translator/Editor.
SPELL Interpreter: based on one `call_proc` predicate.
3. **EPOS PM tasking framework, using SPELL.**
Task type = activity rule, with root `TaskEntity` defining:
Dynamic PRE–CODE–POST,
Static PRE–POST, FORMALS and DECOMPOSITION.
Planner instantiates, **Execution Manager** enacts,
both in Prolog and using XPCE graphical package.
4. **Project-specific types, instances, and tools.**
E.g. Family with Interface/Body, Decomposition, Dependencies.

Layers 1–3 are **EPOS PM Framework**. On Sun-4 and X.

4.2 EPOSDB and PM application

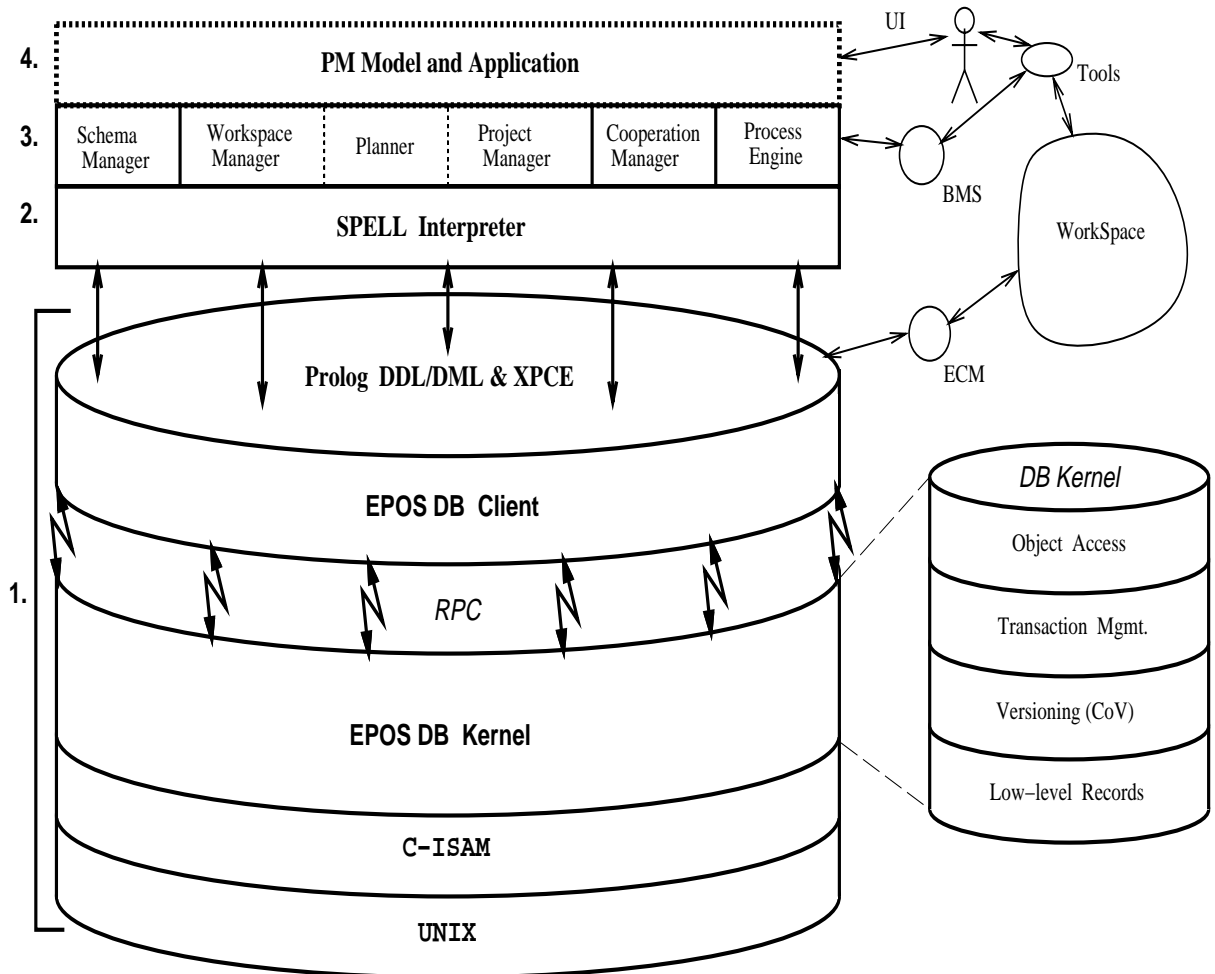


Figure 1: The EPOSDB and client architecture.

4.3 EPOS distributed architecture

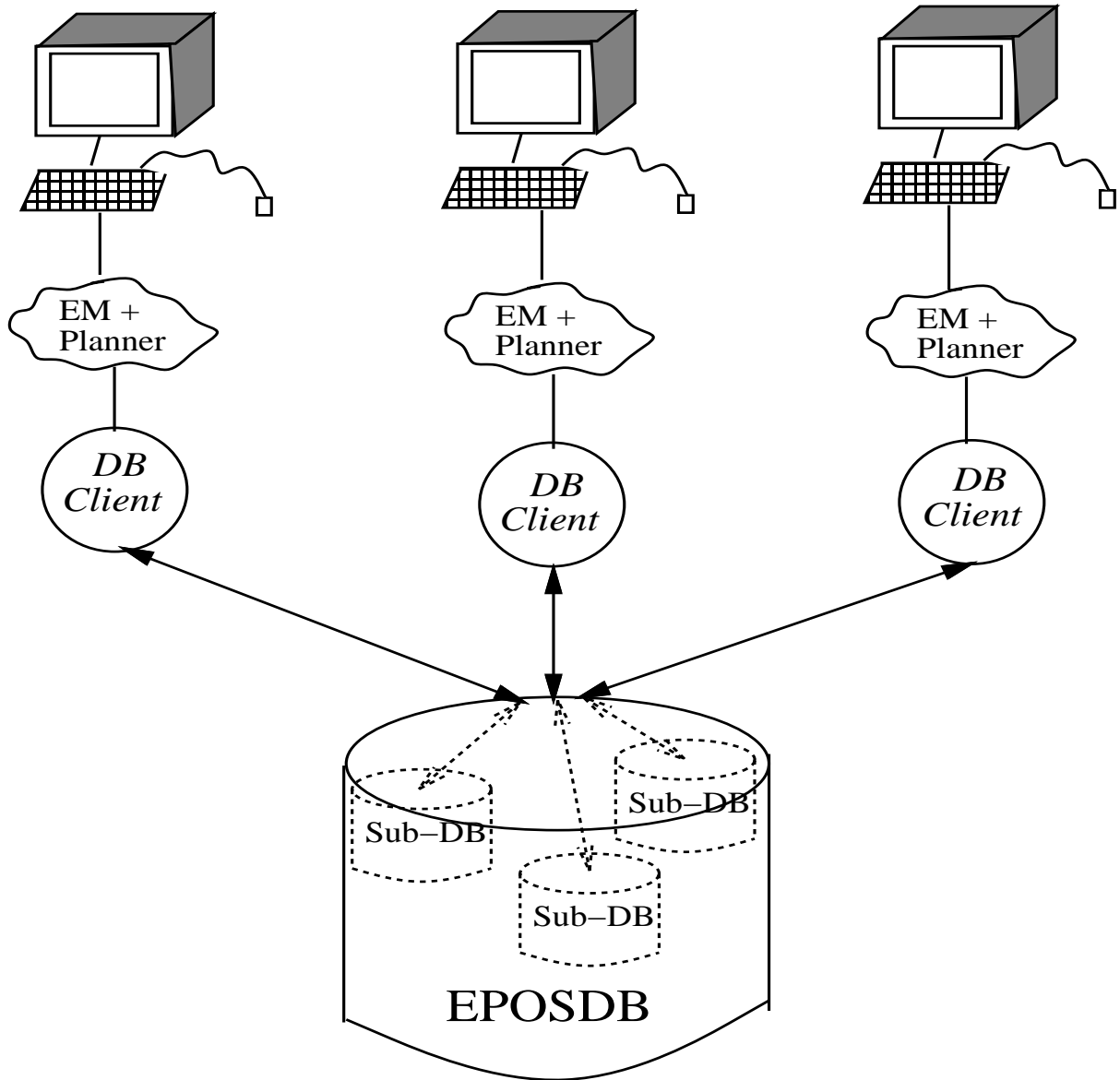


Figure 2: EPOS distributed architecture.

4.4 EPOS process model architecture

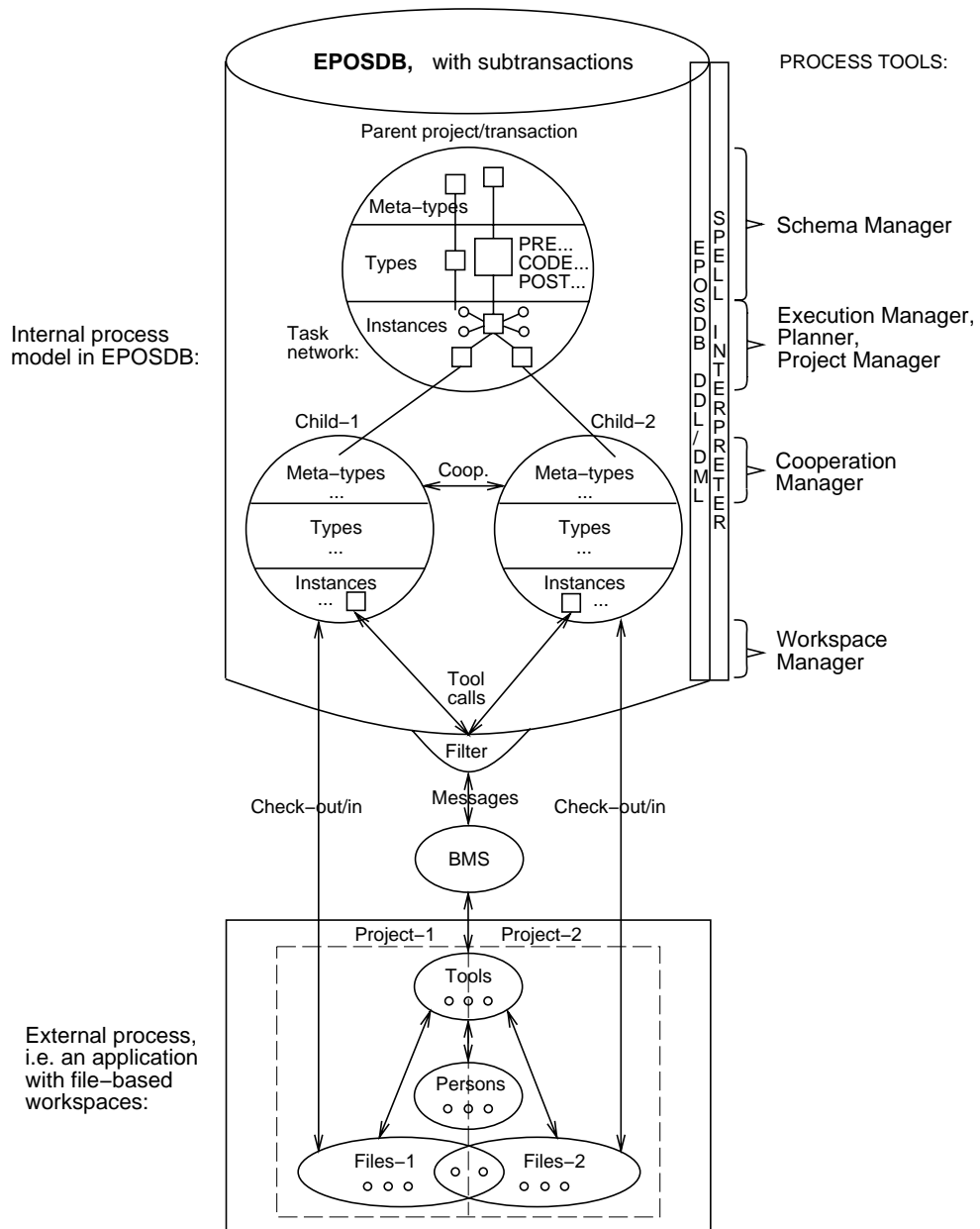


Figure 3: EPOS process model architecture, with indicated process tools.

4.5 ISPW7 reference example in EPOS

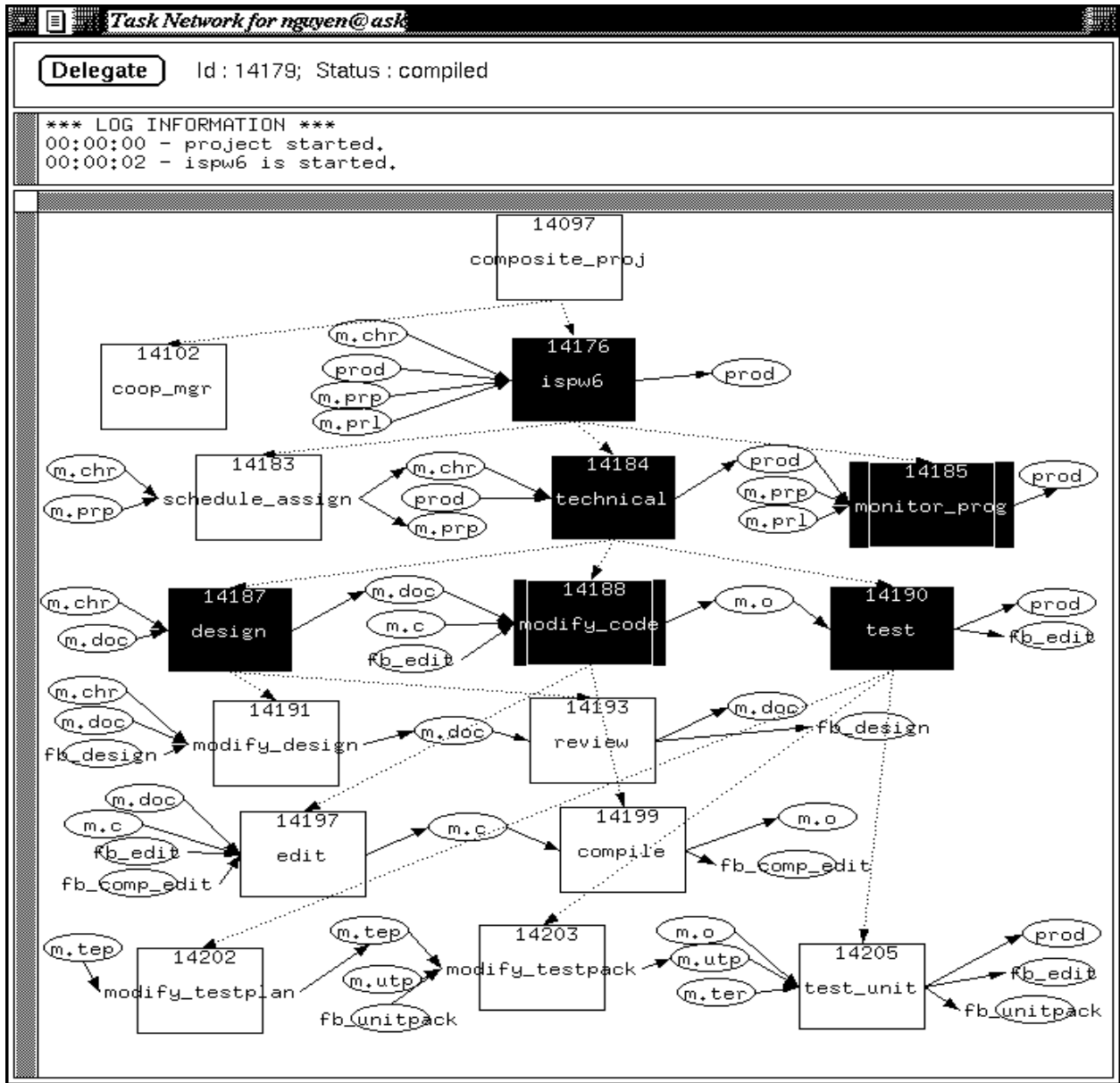


Figure 4: ISPW7 example.

5 EPOS Model Architecture

Model fragments in an EPOS process model:

1. “Classic” process model: Set of instances (objects and relationships) and their schema (types and meta-types), all in **SPELL**.
2. Versioning model information: stored options and rules, selected choice and ambition.
3. Transaction model information: transaction tree, locks, access paths, cooperation protocols.
4. Textual descriptions to express: project, version, product, and transaction information in a Transaction Description Language.
5. Temporary graphical descriptions / views of all above.
6. Misc. secondary descriptions: BMS tables, ...

Underlying EPOSDB offers uniform versioning of 1–4; one view of work-context per transaction/project.

EPOS process model: basically a typed network of chained and decomposed activity (*task*) instances.

Submodels for the following **process elements** will be presented later:

1. BA, Basic ones: activity, product (artifact), human.
2. PR, Project aspects: organization.
3. US, User support: workspace, tools, interop., UI.
4. VA, Variability: support for evolution.
5. Others: versioning, cooperation, quality, ...

5.1 The EPOS PML: SPELL

SPELL: Software Process EvoLutionary Language [C⁺92]:

- OO extension of the Prolog DDL/DML of EPOSDB, to better describe task dynamics and model evolution.
- The extensions include:
 - Procedures and their parameters.
 - Triggers on the procedures.
 - Type level properties (indirectly provided also by EPOSDB DDL, but explicit in SPELL).
 - Some associated semantics: inheritance and scoping.
- Instance- and type-level information specified in two distinct sections inside a type definition (as Smalltalk). Only one meta-type for tasks: type-level info hangs on TypeDescriptors.
- Procedures are inherited by subtyping; redefine bodies.

Ex. Some of the EPOS Predefined Types:

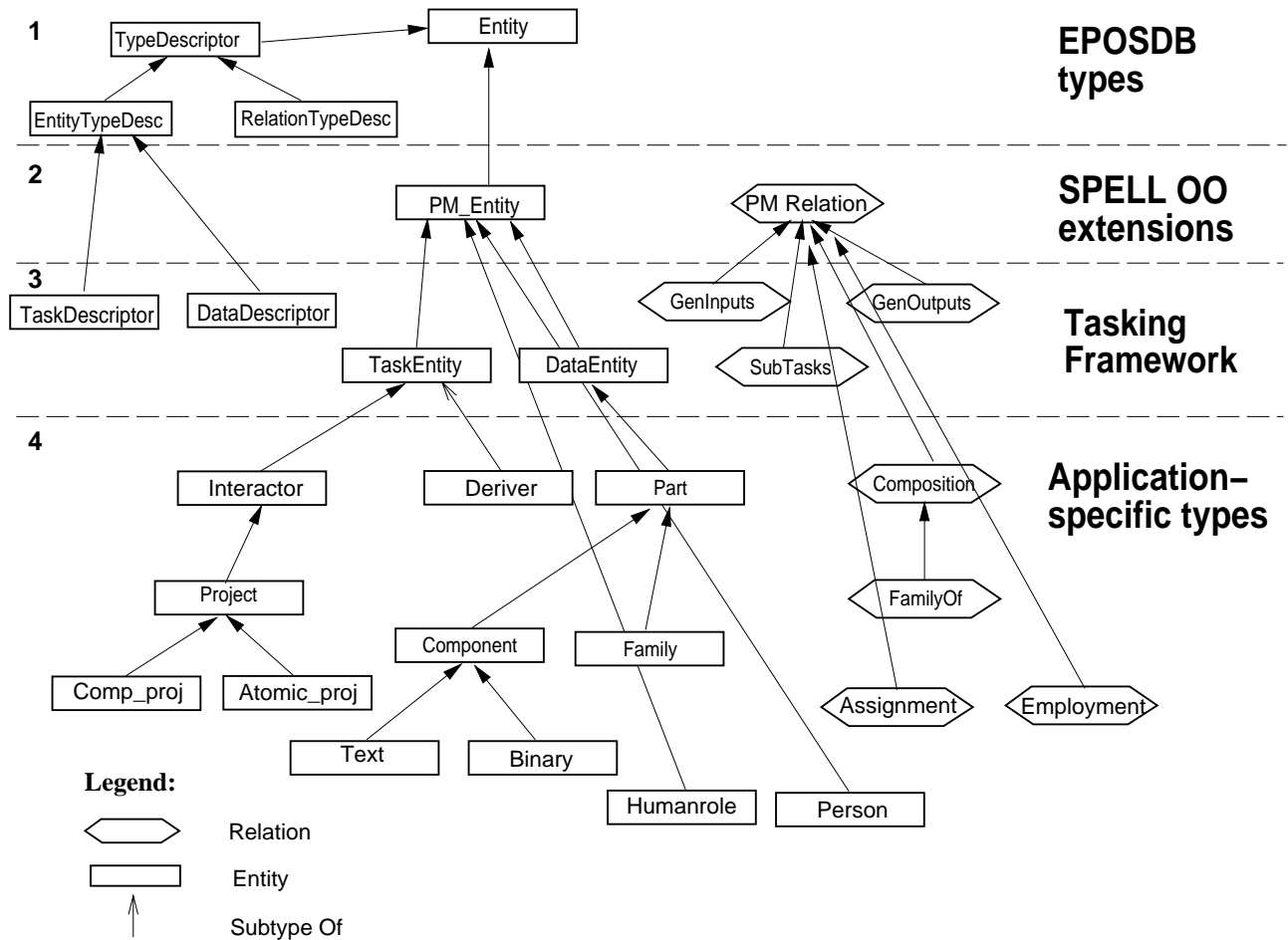


Figure 5: Layered meta-types, types, and instances.

The previous architectural layers are indicated on the left.

Ex. SPELL for task type called **design**:

ENTITY_TYPE Design: TaskEntity {

INSTANCE_LEVEL

ATTRIBUTES

Exec-state: ... := ...; % For task implementation.
 TaskState: String := 'Wait—Active—Term' % Similarly.

PROCEDURES

i.change(New prop.) = ...; % See under Schema Manager, Sec. 6.6
 i.delete() = ...; % Delete - Destructor
 i.convert(NewT:TID) = ...; % Soft change
 start() = ...; % see in
 restart() = ...; % 6.6
 stop() = ...; % similarly.
 makegoal(Goal) = ...; % Issue a goal for the Planner when building
 % sub-network. Is empty due to atomic task.

TRIGGERS

ON-PROC = ... WHEN = AFTER

COND = ... ACTION = ...

TYPE_LEVEL

ATTRIBUTES

PRE_STATIC (Inher=append) Pred = Requirement.status is created;
 PRE_DYNAMIC (Inher=append) Pred = Input.state is satisfied;
 CODE (Inher=concatenate) Prog = gather data; invoke design tool; ...;
 POST_STATIC (Inher=append) Pred = DesignDoc.status is edited;
 POST_DYNAMIC (Inher=append) Pred = true;
 FORMALS (Inher=redef) String = 'in:\$Requirement(read)⇒
 out:\$DesignDoc(write)';
 DECOMPOSITION (Inher=redef) String = 'REPertoire(Task1, Task2, ...)';
 EXECUTOR (Inher=redef) String = '<DesignTool and parameters>';
 ROLE (Inher=redef) String = '<Logical role name, e.g. Designer (see in 5.4);>'

PROCEDURES

i.create(NewTask:TID) = ...; % Normal NEW generator - Constructor
 subgoals(...) = ...; % Used by Planner in sec. 6.5
 % to generate a goal for each its inputs.

} % Design task type

Figure 6: The Design task type.

5.2 BA.ACT: Activity or Task Submodel

Task instance = active DB object, in *task network* (cf. “Petri net”), incrementally constructed/planned, concurrently enacted.

Task type = *activity “rule”*, with a script to be executed.
As a subtype of predefined `TaskEntity` type.
Two main subtypes: Interactors and derivers.

A task type expresses knowledge about a development step/tool, and contains:

- **Dynamic** PRE- and POST-conditions for dynamic triggering of tasks, executing between a sequential `CODE` script. Side-effects allowed on such PRE, `CODE` and POST.
- **Static** PRE- and POST-conditions for forward or backward reasoning without executing the `CODE`.
- `FORMALS` and `DECOMPOSITION` type constructors regulate structure of task network.
I.e. network templates or a minimal **graph grammar**.

Ex. Task Network:

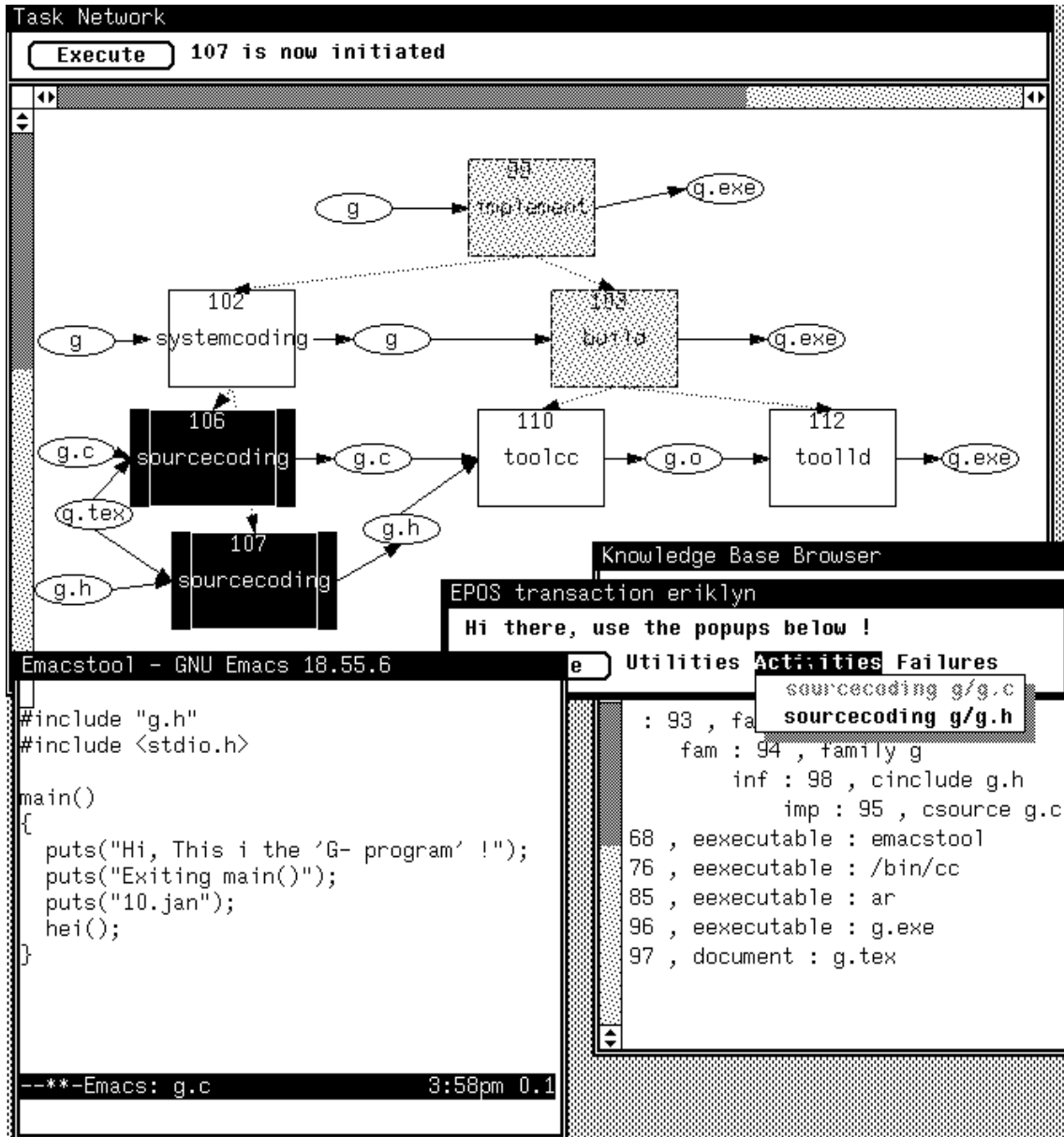


Figure 7: Simplified snapshot of EPOS-PM to produce g.exe.

5.3 BA.PROD: Product submodel

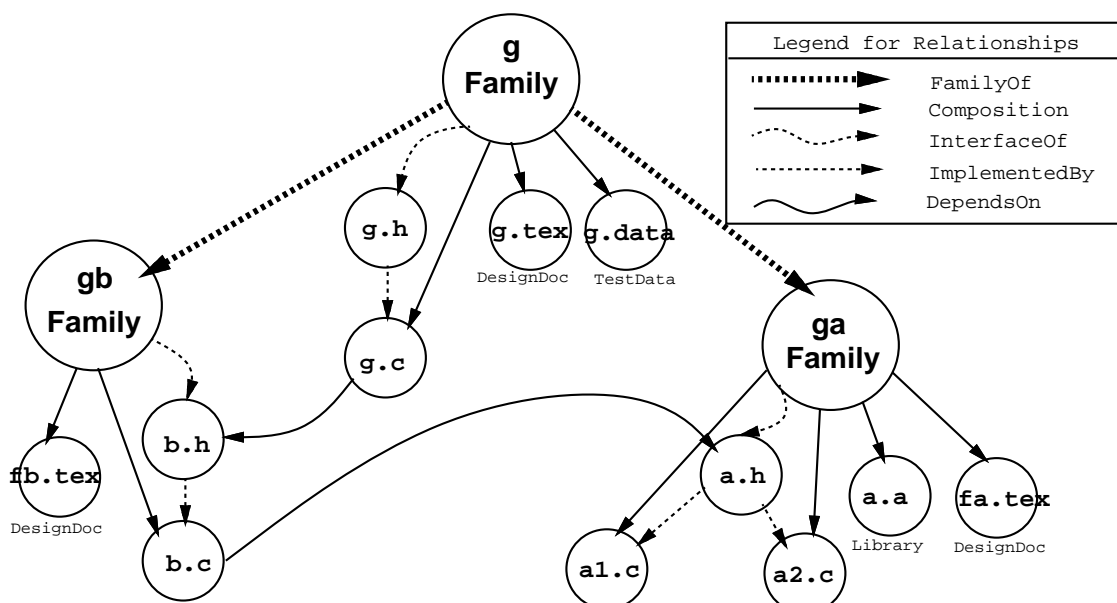
Product: passive objects, connected by relationships
(task = active object).

Product type: here defined in an OO ERA formalism (OMG-like), extended with procedures and triggers.

Pre-defined product models available for certain domains
(p.t. software engineering), and supported by tools.

A simple, textual **product description language (EPDL)** is used to describe products in the workspace, a mini-MIL.

Ex. Product families with Interface/Body, and with `Part_Of` and `Depends_On` relationships.



5.4 BA.HUM: Human/role submodel

Rudimentary modeled:

- **Role:** generic person (“placeholder” for an activity), with certain rights and plights. E.g. a Manager, Designer or Programmer.
Connected to the ROLE attribute in tasks.
- **Agent:** concrete person filling a role, thus responsible for specific objects or operations. E.g. “Donald Knuth” or “Bill Curtis”.
- **Team:** set of agents.

5.5 PR.ORG: Organization/project submodel

Project is the EPOS term for execution environment – multi-user or single-user, large or small:

- Project is high-level task to control sub-DBs / workspaces, unit of customization / evolution.
- Connected to an EPOSDB transaction.
- Contains:
 - General information: name, responsible, informal work description and change log.
 - Work order or modification request.
 - Product description (in EPDL), and version description.
 - Workspace description.
 - Cooperation protocols (see 5.12).
 - Resources: time, humans, HW/SW tools.
 - Constraints: deadlines, quality goals, own meta-model.
 - Subprojects.
 - ...

All described in a textual **Work Unit Description Language, WUDL**.

Strong overlap with project management!
No general organization modeling.

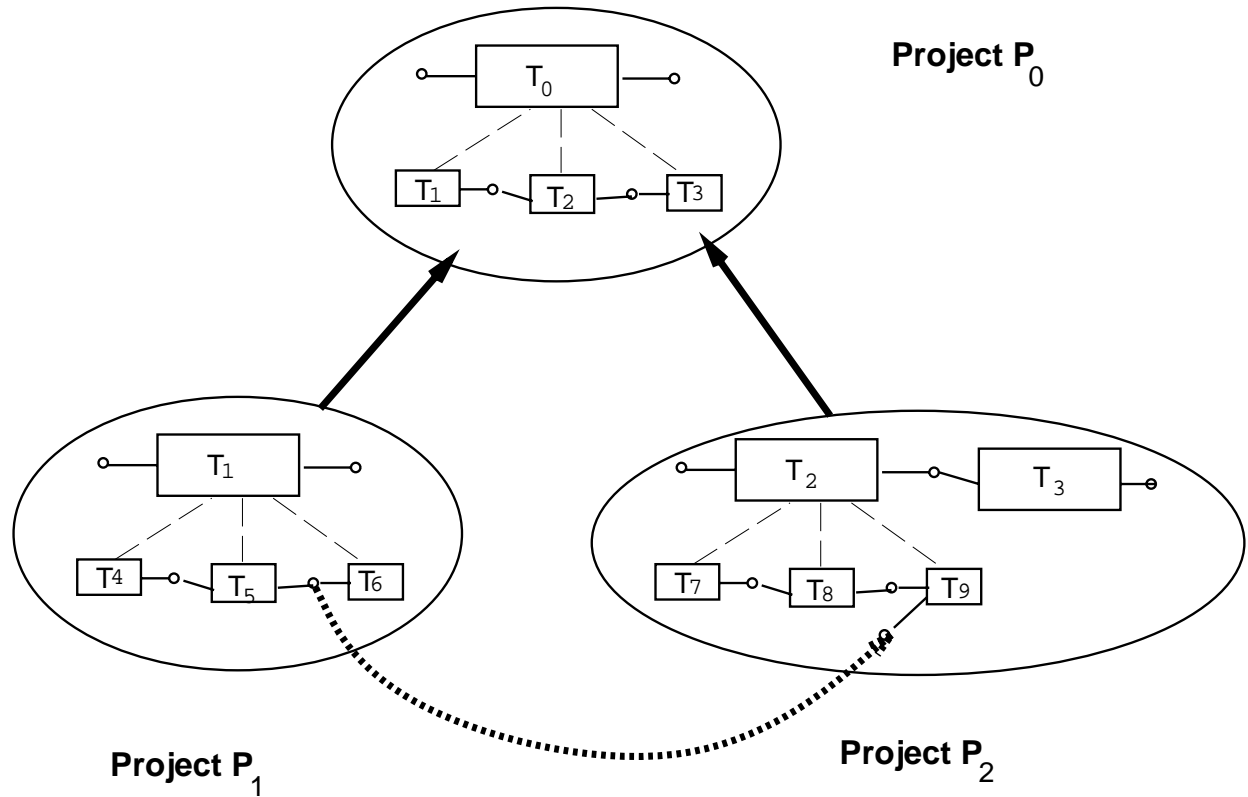
Project breakdown: Task partitioning and data exchange:

Figure 8: Project breakdown: Task partitioning and precommit exchange of data.

5.6 US.TOOL: Tool submodel

Tool instance: passive object of type Executable, “external” PROC with (file) name. A proxy, not the OS-tool itself.

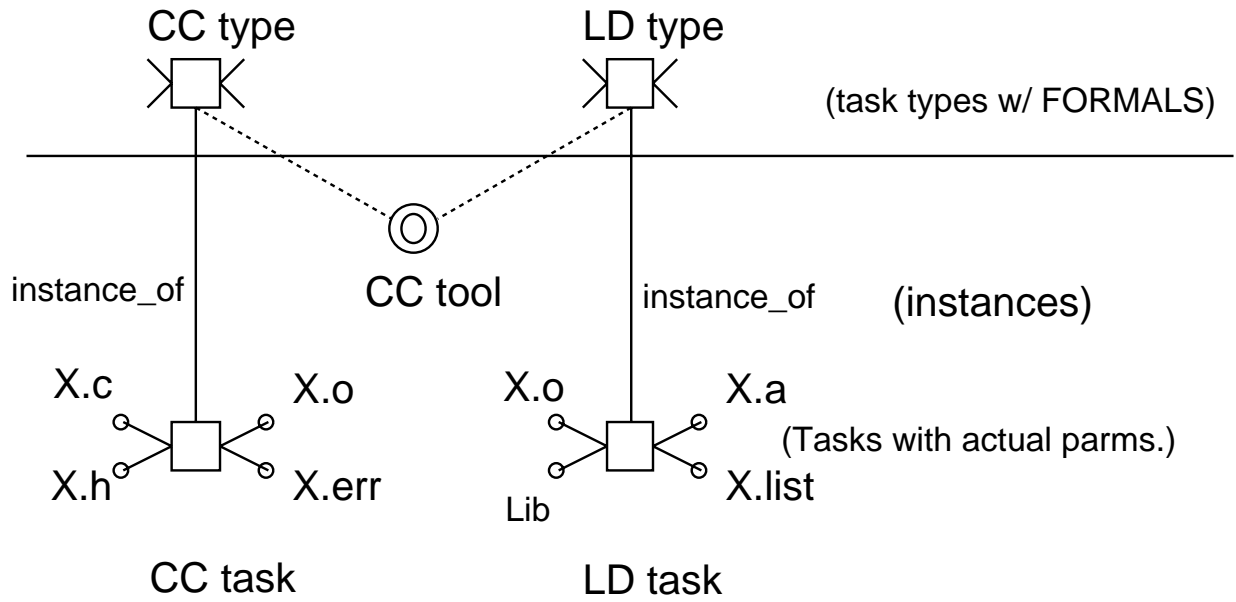
OS-tool: interactive (e.g. editor) or derivative (e.g. compiler).

Tool task subtype:

- Generalizes and *describes* an external OS-tool.
- CODE: *script* to activate and to control tool invocation.
Pre/Post-Actions as *envelope*.
- Associated OS-tool: via EXECUTOR attribute to Executable object, pointing at an independently produced and versioned OS-tool.
- *OS-tool switches*: very special task “parameters”.
Some may change the FORMALS, cf. Unix CC compiler!
I.e. *several* deriver subtypes (“variants”) for *one* OS-tool.

Tool invocation: by a networked task instance, cooperating via a BMS against an external and concurrent OS “thread” that executes the given OS-tool.

Ex. Tools coupled to tasks:



5.7 US.WS: Workspace submodel

Very simple **DB-WS mapping** of internal product structures 1:1 down to Unix files. Allows shared families. Described in TDL.

5.8 US.INTE: Tool Integration submodel

EPOS uses simple shell scripts and manually filtered messages for asynchronous invocation and communication with external tools. No explicit model for this.

BMS (Broadcast Message Server) configuration schemas must be maintained separately.

5.9 US.UI: User Interface submodel

EPOS has no systematic model here, mostly ad-hoc.

The relevant models are displayed both graphically and textually by their manipulating PSEE tools.

Agenda attached to the Execution Manager.

5.10 VA.EVOL: Meta-process submodel

Explicitly stated by meta-types and related meta-activities and -procedures in SPELL [JC93], often as a separate task network. Meta-activities include basic mechanisms (e.g., versioning, subtyping) for evolving types by using reflective SPELL, dynamic OO binding, versioning in subprojects.

Meta-process covers four major functional aspects:

- **Planning:** prescribing how a given project is scheduled and then configured into an appropriate development process (cf. Planner and Project Manager).
- **Tracking/follow-up:** guidance on how to collect relevant data/measures on which historical profile/experience will be compared and analyzed. Effective directives and guidelines are fed back to the project environment for corrective actions (cf. Project Manager).
- **Packaging/learning:** describing how experiences and lessons gained from completed projects are generalized, formalized and stored for future use (preventive actions).
- **Evolution:** describing how the entire software process (p-p, m-p and p-s) is evolved (cf. Schema Manager and new Task Network Editor).

Tool support (meta-tool):

- **(Re-)Planner**: incrementally (re)constructs task network based on an initial goal and current world state.
- **Project Manager**: some low-level aspects for project management.
- **Execution Manager**: interprets the task network and assists in task execution.
- **Task Network Editor (new)**: changes task network to accommodate to specific needs (changing-on-the-fly).
- **Schema Manager**: provides support for updating types and schemas.

5.11 VERS: The versioning submodel

Traditional versioning

- Sequential revisions. May e.g. be used for:
 - Corrections
 - Adaptations
 - Enhancements
- Parallel variants - may be permanent or temporary.
- Merging - combining variants. Problematic.
- Delta storage — to save space and record differences.
- Attribution — alternative way to select versions.
- Change sets: collection of changes explicit.

Change Oriented Versioning – COV:

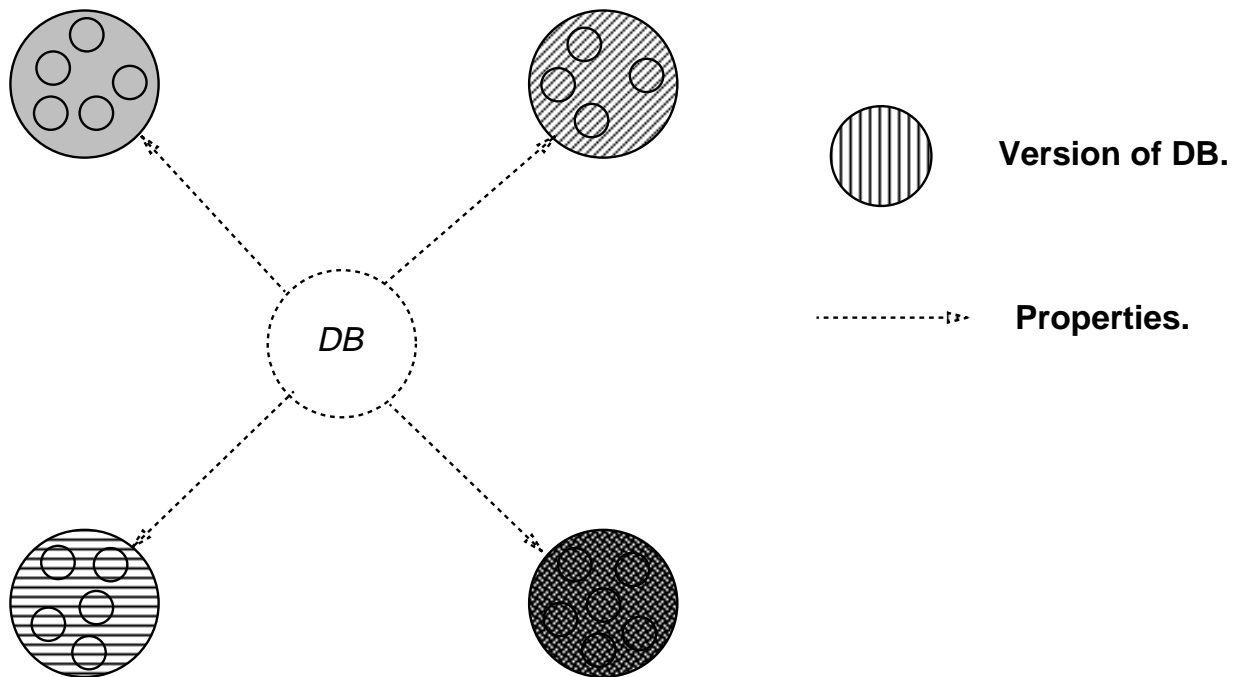


Figure 9: Versioning of the database.

- Versions are identified by their logical properties, not version ID.
- The DB is versioned as a whole.
- No strict revision chaining.
- Raw merge comes for “free”.
- Versions are *implicit*; result of selecting among properties.

See [Lie89] [MLG⁺93].

COV (cont'd 1):

- Functional change, primary: Σ updated fragments = “deltas”.
- Version, secondary: accumulated Σ changes.

- A version is not an explicit object, but the result of a version selection/evaluation.
- COV is largely independent of data model, and enables uniform versioning of entities, relationships etc.

COV offers:

- Intentional and flexible version selection, based on logical changes.
- Explicit representation of version space (for locking?).
- Versioning being orthogonal to data and its granularity.
- Simpler version selection: done once, not for every object.

COV (cont'd 2):

- **Option:** (global) boolean variable defining one logical property.
- Can be freely combined, possibly constrained by stored version-rules.
- **Choice:** for selecting version – binding of true/false to relevant options.
- Defines single-version view of the whole database *before* product selection.
- **Ambition:** set of bindings, like the choice – defines scope of changes being done.
- All versions “within” ambition are affected.
- **Version rules:** a *constraint*, a *preference*, or a *default*, used for guiding or restricting setting of ambitions and version-choices.
- Status properties can be associated with versions, ranging from “untested” to “released-and-frozen”.

Configuration Mappings on top of COV

VD, Version-description (+ Preferences and Defaults).

PD, Product-description = (Root objects, ER types).

CD, Config-description = [VD, PD].

CD \mapsto *bound* Configuration, in three steps:

- C1) VC = Bind-choice(VD, version-rules).
- C2) Sub-DB = Version-selection(VC, DB).
- C3) Configuration = Product-selection(PD, Sub-DB).
- C4) Workspace = Check-out(WS-DB-map, Sub-DB).

Product (“AND”) selection *after* version (“OR”) selection!

Configuration: consistent, complete version of process model;
more than sub-product!

EPOSDB mappings

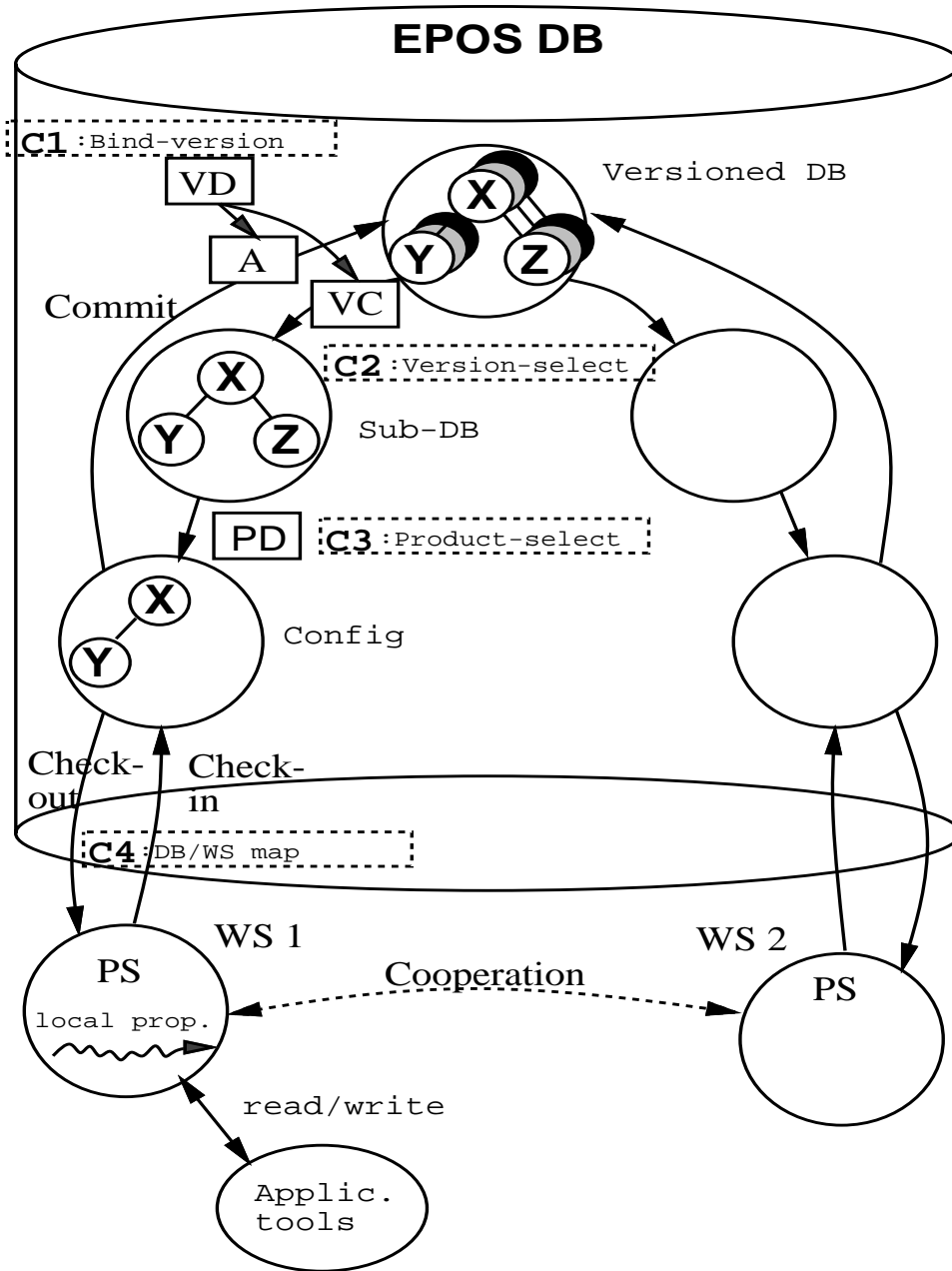


Figure 10: EPOSDB mappings.

5.12 TRANS: The transaction submodel

EPOSDB transaction model [CM91] [NC93] [LL95]:

- A hybrid between strictly nested DBs and shared blackboards.
- **Nested, non-serializable, long** transactions, with version-choice (for read) and ambition (for write); product modeling done by EPOS-PM outside EPOSDB.
- **Cooperating transactions:** May cross-propagate to and negotiate with siblings *before* commit, according to agreed-upon protocols.
Own EPOS message facility.
- Flexible access locks and access paths.
- Also short, ACID transactions for critical regions.
- Transactions as DB-objects and rel.ships.
- Local changes in sub-DB, then **commit** upwards to parent (which may propagate downwards to remaining children).
- Flexible commit / abort / merge policies, with user-defined consistency constraints and conflict resolution.
- Basic mech. in EPOSDB, domain policies in EPOS-PM.

- External file-based **workspaces**, with check-out and check-in.
- Transaction: **work context** for customized process model, with (meta-)types, task networks, and product descriptions.
- Coupled to EPOS projects to define work environment. E.g. project's POST-condition defines commit consistency.

Work-Units

- **WUDL** – A language to specify planned activities/transactions. Specifications stored in the EPOSDB, and can be changed dynamically.
- Express work intentions within a workspace, and cooperative behavior.
- A work-unit specification includes:
 - Name (hierarchical), Description, Product
 - Read- and write-sets. Corresponding lock and access modes.
 - Cooperation policies: Responses to notifications.

Overlap and Propagation

Transactions may **overlap** and produce conflicts — need procedures (merge, cooperation) to handle that.

Version-overlap: mutual version visibility; ambitions overlap if no conflicting option setting.

Product-overlap: shared subsystems; non-empty intersection of actual product components.

Policies for handling conflicting updates:

- *Rollback* (prevent any overlapping transaction from committing – intolerable!).
- *Priority* (e.g. let the first or last “win”).
- *Access locks* (controlling Read/Write by 2PL).
- *Optimistic* (“soft locks”).
- *Merging/integration* (most general, used in EPOS):
either continuous cooperation or pre-commit merge.

Ex. Parent transaction **T**, with **N** children **T_i**.

T_i works on disjoint product parts.

T_i hides (private) updated modules **M_x** in local Workspaces; will gradually “pre”-check-in (promote) these to shared Workspace pool + ensuing propagation into other Workspaces.

What if **T_j** wants to *delay* using a promoted modules **M_x** from **T_i** ?

- Temporarily keeping a local copy of the previous version of **M_x**,
- By equivalent manipulation with symbolic links or search paths.

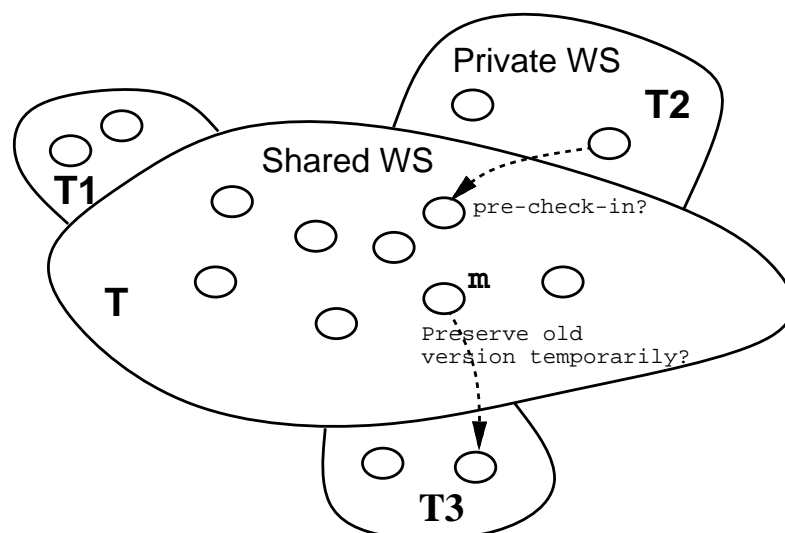


Figure 11: Partly shared Workspaces, same ambition.

Change propagation between transactions:

- Decide mutual version visibility or overlap.
- Set up (“plan”) detailed and pairwise **inter-transaction** protocols for pre-commit negotiation and propagation:
 - **What**: limit propagation to certain datatypes, to explicitly mentioned objects, ...
 - **When to receive** (and implicitly to send) changes.
 - * **BUSY**: all changes are propagated immediately.
 - * **SEMI-BUSY**: Manual confirmation or request.
 - * **SELF-COMMIT**: Made visible after transaction commit.
 - * **OTHER-COMMIT**: integrate changes upon other’s commit.
 - **How to accept**:
 - * **NEEDING-ACK**: need explicit ack after notification.
 - * **NO-ACK**: notification always sent, no answer expected.
 - **How to copy** (Workspace connectivity):
 - * **AUTO-COPY**: shared file (only for NO-ACK), etc.
 - * **MANUAL-COPY**, e.g. to prepare for merging.
- **Planning support by TRAPLAS (6.7)**:
partition, schedule and protocolize subtransactions.

5.13 General Model Organization and Evolution

Model structuring:

- General instances with OO types/meta-types,
- Type inheritance,
- Instance-level composition for product/activity/project,
- Type-level composition by network templates (FORMALS, DECOMPOSITION),
- Sub-projects with versioning,
- Reusable submodels, subschemas ??

Model customization and evolution:

- Reflection by meta-types: but non-SPELL submodels.
- OO dynamic binding /polymorphism,
- Universal versioning (PM+CM),
- Copy and overwrite.

6 EPOS Tool Architecture

6.1 General architecture and examples

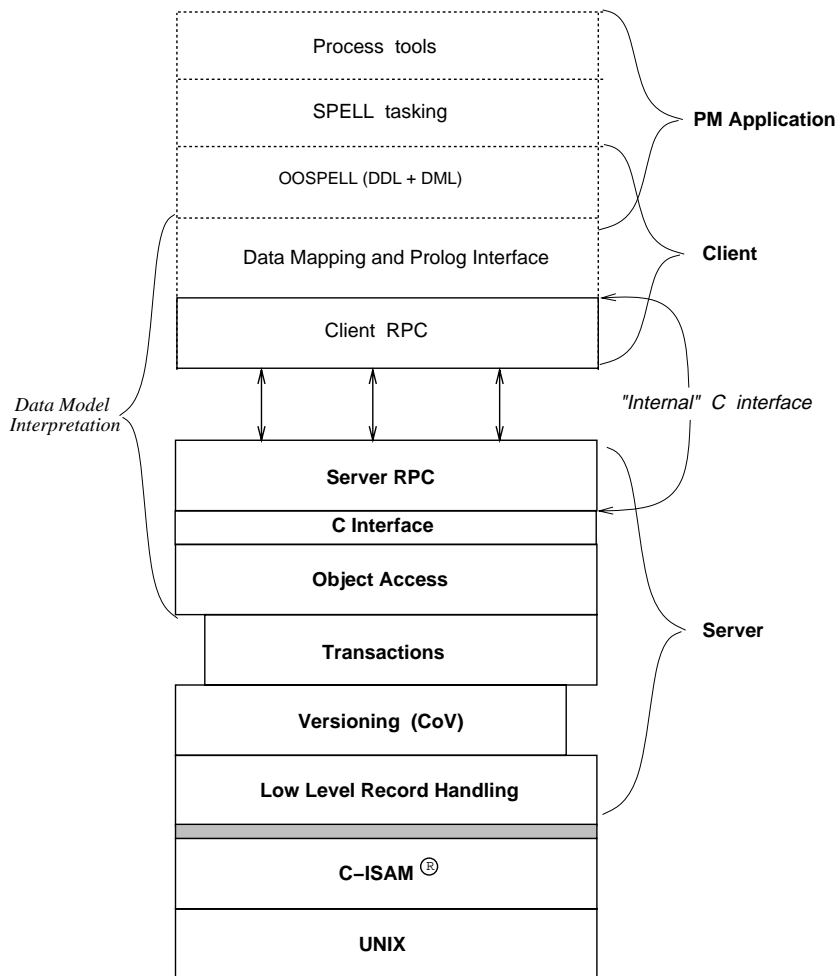
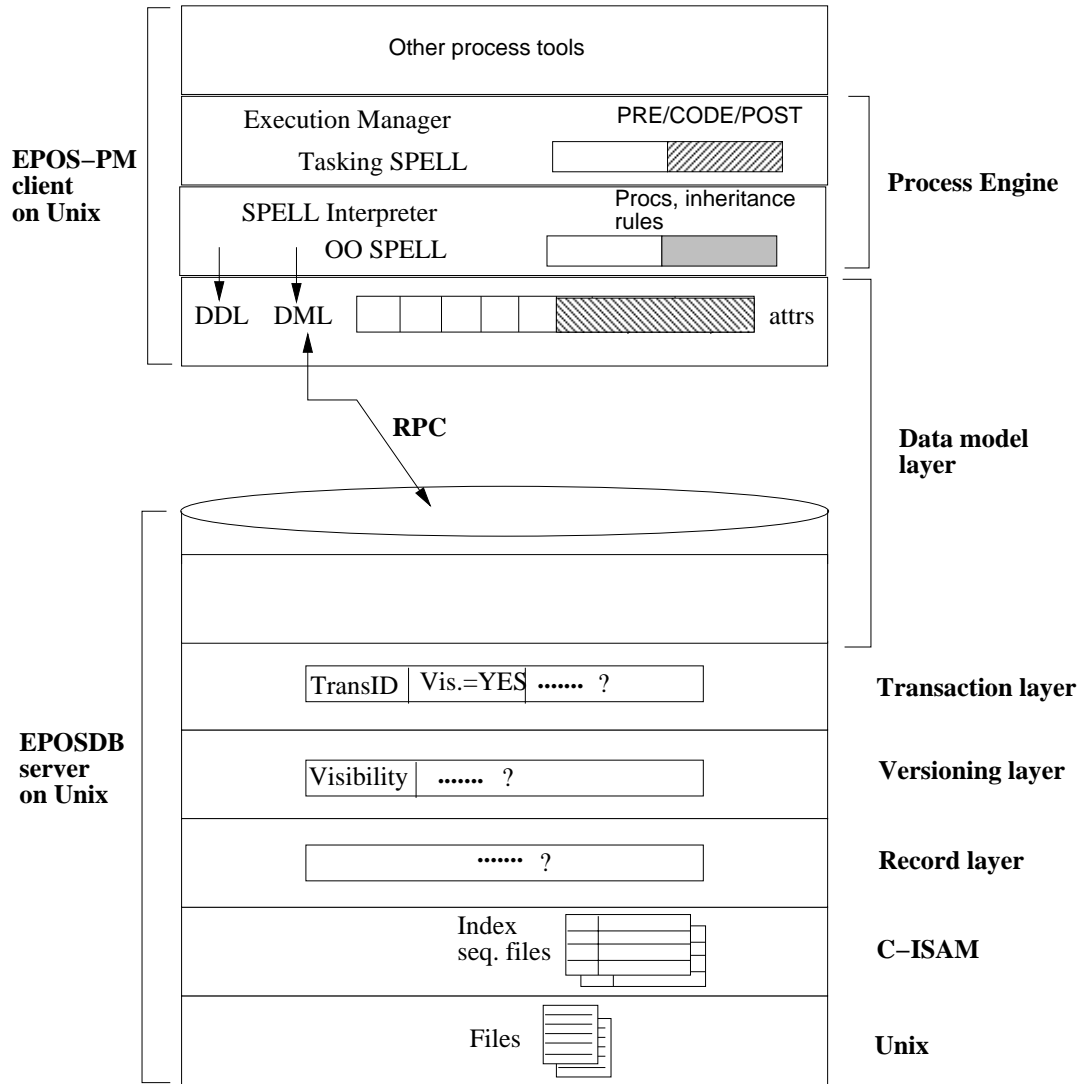


Figure 12: The layering of EPOSDB and EPOS-PM.

EPOS layers and data:



In EPOSDB lies also schema (TypeDescriptors), versioning options, and transaction descriptors.

Figure 13: EPOS layers and data.

6.2 EPOSDB

Client-server, versioned software engineering database, offering a DDL/DML in Prolog.

Architectural or functional layers:

1. PSEE as client application on top of EPOSDB:
 4. General process tools.
 3. Tasking SPELL, with Execution Manager.
 2. OO SPELL, with SPELL interpreter.All above: prev. layers **4, 3, 2**.
All below: prev. layers **1**.
2. Data model layer: in both client and server, using RPC.
3. Transaction layer.
4. Versioning layer.
5. Record layer.
6. C-ISAM, index sequential files.
7. Unix, flat files.

Data, versioning and transaction model/layers: **independent!**

SPELL extends EPOSDB data model with:
full OO of DDL+DML, reflection, and tasking.

6.3 SPELL Interpreter: Low-level Process Engine

Type representation: `TypeDescriptors` (TDs).

Type operations: type-level PROCs & attrs to instrument the PSEE tools.

Access/bind semantics of PROCs, triggers and attributes by *one* predicate:

```
call_proc(Called, Caller, PName, Parameters).
```

Distinguishes between type-level and instance-level PROCs.

For instance-level attributes: use DML predicate names as parameters.

For type-level attributes:

```
call_proc(Caller, Called, read/write,  
          [A_Name--A_Val]).
```

6.4 Execution Manager: Low-level Process Engine

Execution Manager adds tasking and user/tool interface:

- One per transaction, acts like a coroutine.
- Cooperates with users by simple UI, incl. an agenda.
- Cooperates with production tools by a BMS.
- Cooperates with Planner to incrementally (re)generate **plans** = partially ordered task networks.
- Handles **errors** via *error tasks* – exception handling!
- Checks **PRE_DYNAMIC** conditions of tasks;
non-temporal conditions (RULE: timestamp changed),
temporal conditions (POLICY: when to compile).
- **Task states:**
 - *Created*: unfinished – used during planning.
 - *Initiated*: waiting for non-temporal PRE conditions.
 - *Waiting*: waiting for temporal PRE conditions.
 - *Ready*: all PRE conditions fulfilled; waiting for re-sources.
 - *Active*: execution of the **CODE** part.
 - *Terminated*: execution finished.
- **Goal-directed** (lazy, the classic *Build*) and **Opportunistic** (e.g. busy) execution; *task-specific*.

EPOS task network internals:

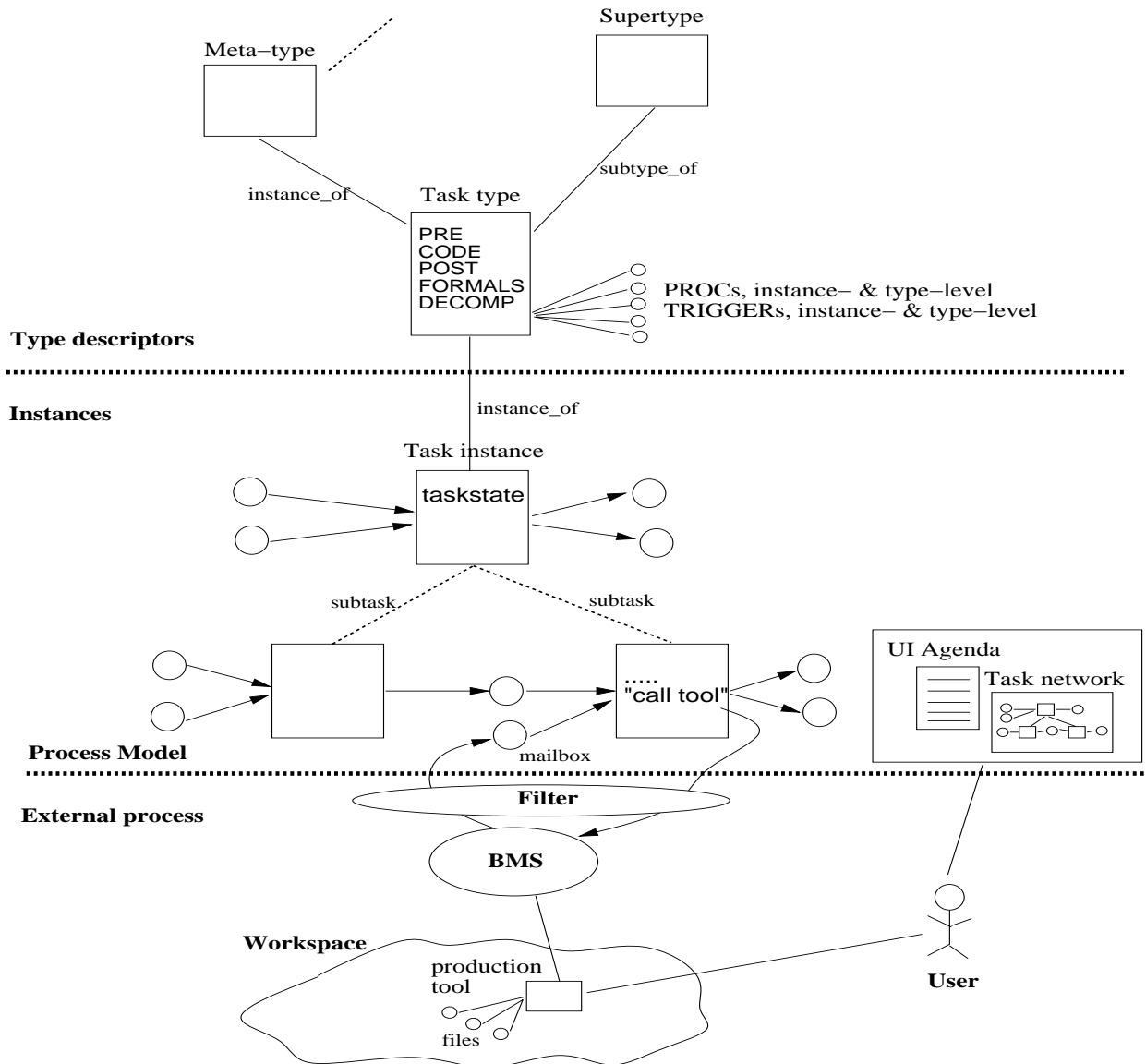


Figure 14: Task network and typing structure.

BaMSE – a BMS connection:

- Student group project autumn 1993.
- Integrate with the BMS (Broadcast Message Server) of the HP SoftBench environment.
- Execution Manager runs tools by sending messages via BMS.
- Messages sent back upon certain events (e.g. tool termination).

Similar to Adele [BEM93], Process Weaver [Fer93], SPADE [BFG93] etc. using BMSes like HP SoftBench, DEC FUSE.

Execution Manager, BMS and production tools:

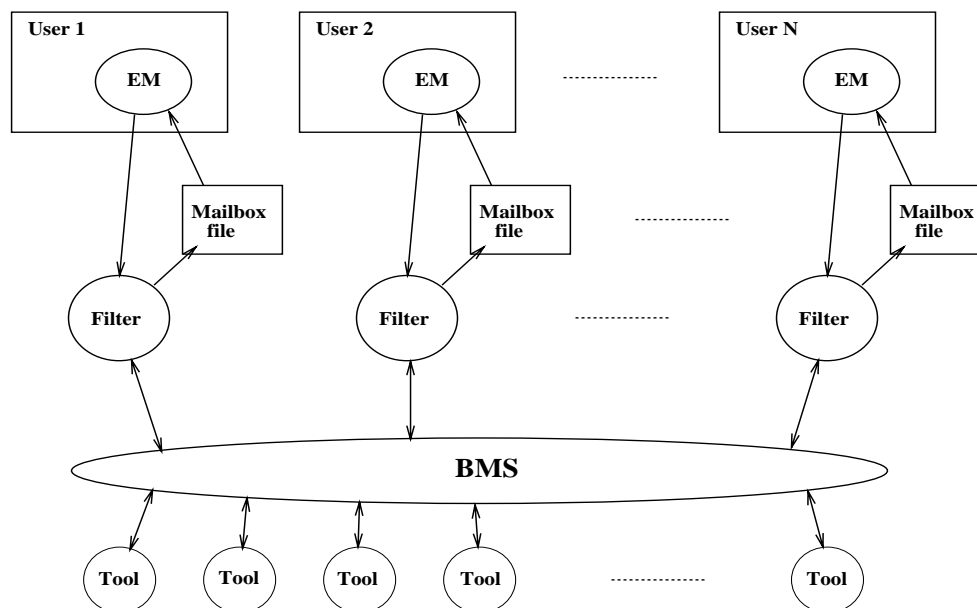


Figure 15: System architecture with BaMSE in use.

6.5 Planner

Planner incrementally (re-)generates task networks [Liu91] [LC93]:

- Cooperates with Execution Manager: receives **goals** and returns **plans** for execution.
- **Product-level**: e.g. *horizontal* derivation graphs.
- **Project-level**: e.g. simple task *breakdown* (see Project Manager for complex planning by humans).
- **Replan** upon execution failures, i.e. unexpected plan results, and upon product changes.

Planner internals:

- **World State Description**: TDs and Product.
- Reasoning on **PRE_STATIC** and **PRE_POST** conditions. Ignores dynamic information (time-stamps, start-up).
- **Non-linear** planning: using **FORMALS** and domain knowledge in a production system.
- **Hierarchical** planning: Expand task with empty **CODE**, use **DECOMPOSITION** and desired inputs/outputs.
- **EPOS task types** (*actions*): explicit I/O, and **FORMALS** translated to normal PRE/POST conditions.

6.6 Schema Manager

The **Schema Manager** handles the type library and its evolution:

- SPELL Translator and SPELL Editor/Browser:
inserts new and modifies old types.
Uses a graphical editor (BEATE student project).
- SPELL Analyzer: assesses proposed type changes.

Type-level PROCs subtyped to instrument enhance SPELL Translator/Editor:

- `t_create`: depends on various consistency checks.
- `t_delete`: OK, unless being referred to.
- `t_change`:
 - *Hard* changes imply structural changes;
illegal, but may use `i_convert` to refine (sub)type.
 - *Soft* changes are PM-specific; e.g. changes in
CODE, dynamic PRE/POST, FORMALS, or PROCs.
Delicate conversions and synchronizations!

6.7 Project Manager

Manages projects, being high-level tasks attached to transactions, in dialog with humans:

- **Start Project:** The parent project/transaction must often delegate to subprojects. Done through its **Project Manager** subtask and its **TRAPLAS** tool: define partitioning, scheduling, and coordination of new subprojects.
- **Generate Subtasks:** Under the local `Project` task, the **Planner** will generate an infrastructure of subtasks,
- **InitProject:** The new subproject has a config-description, `CD`, that represents its sub-database.
- **Customise own Cooperation protocols** in `WUDL`, also by **TRAPLAS**: negotiation/propagation patterns against overlapping sibling projects may be adjusted.
- **Customise the Process Schema:** The **Schema Manager** is used to refine and adapt the `Process Schema`, that has been inherited from the parent project, into a more specific one.
- continued next page...

- **Check-out:**
Relevant files are checked-out from the database by the **Workspace Manager**
- **Real Project execution:**
 - An `AtomicProject` will in its `Develop` subtask, (re)plan and (re)execute further subtasks.
The **Cooperation Manager** is used here.
 - A `CompositeProject` will in its `Project Manager` subtask recurse from above.
- **Check-in:** The **Workspace Manager** will ultimately check-in the modified workspace files to the database.
- **TermProject:** It will commit the current database transaction. The **Project Manager** of this subproject will notify and negotiate with the corresponding **Project Manager** in the parent project about this.

Transaction Planning Assistant, TRAPLAS**EPOS Transaction Planning Assistant [CLH95]:**

1. **Partition** transactions or projects, to limit interconnections between the resulting subtransactions.
2. **Schedule** the above subtransactions, to ensure a reasonably good performance of the overall software process; and
3. **Protocolize** the leaf subtransactions to drive the more low-level database mechanisms, e.g. propagation and negotiation policies, data sharing, and locking.

Uses the following information:

- **Write Set (WS):** Set of objects written (to) by transaction.
- **Read Set (RS):** Set of objects read by transaction.
- **Impact Sets (IM):** Set of objects affected by transaction, directly or via dependencies.
- **Domain-specific knowledge:** product structures, weighted relationships.

Ex. Impact analysis of two transactions:

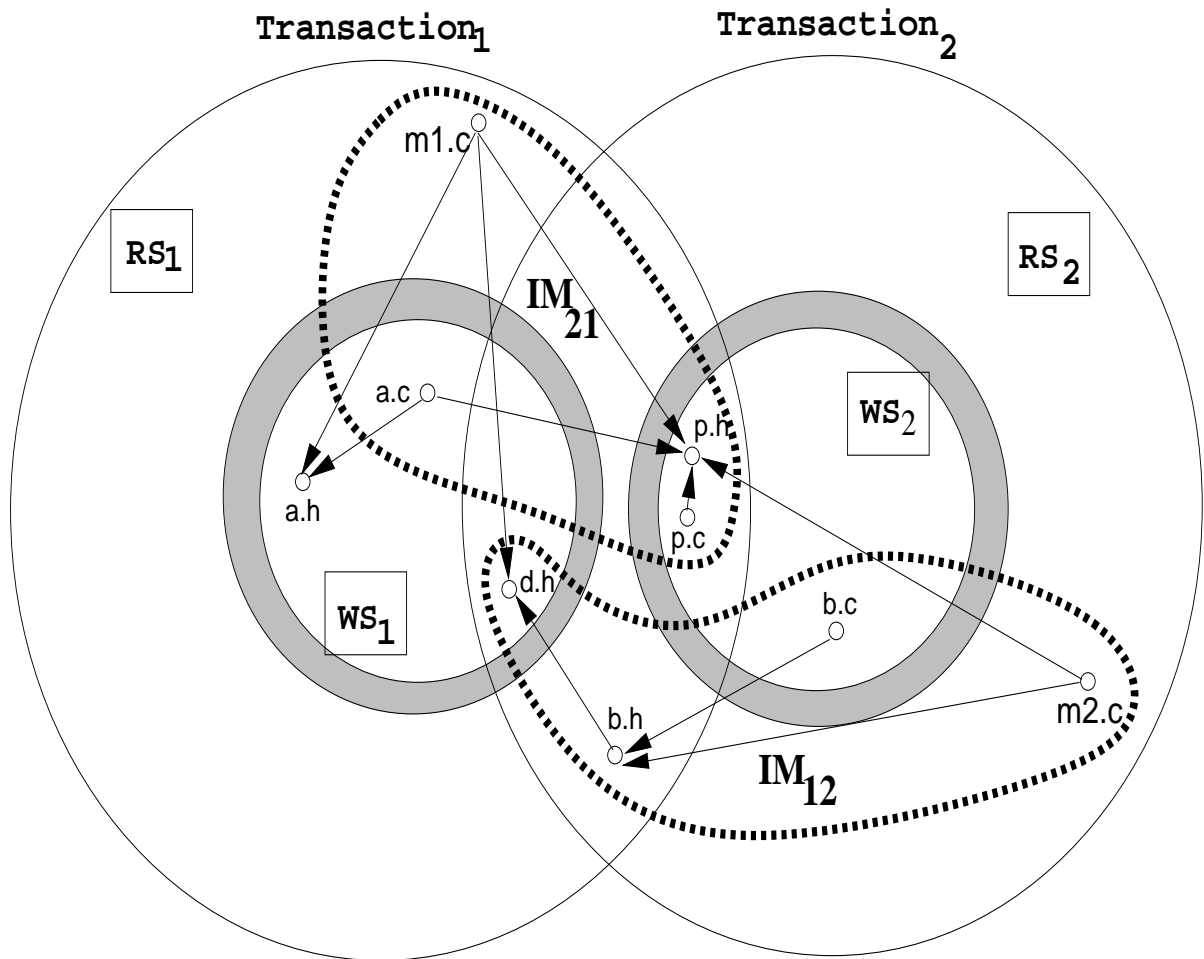


Figure 16: Example: Impact analysis of two transactions.

TC_i – The transitive closure of WS_i (full impact of WS_i)

$IM_{ii} = TC_i \cap RS_i$ – The impact of WS_i on RS_i

$IM_{ij} = TC_i \cap RS_j$ – The impact of WS_i on RS_j

$IM_i = \cup_{j=1}^n IM_{ij}$ – The total impact of WS_i .

6.8 Workspace Manager

This uses the **ECM (EPOS CM)** tool to check in/out single files or groups of such from a workspace against the EPOSDB, using the textual EPOS Product Description Language, EPDL.

Project structure during work:

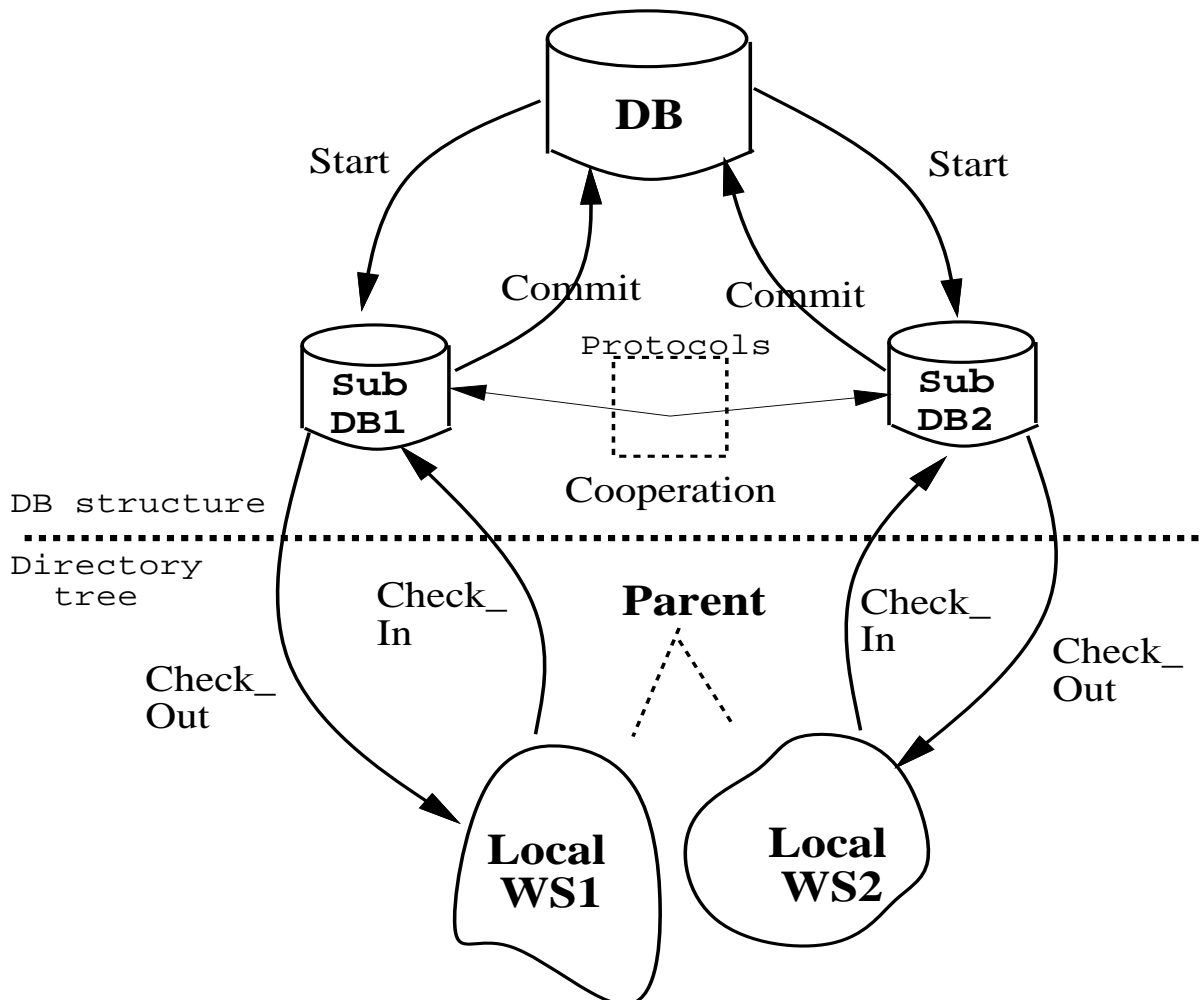


Figure 17: Project structure during work.

ECM has a set of commands to start and close workspaces, component-wise check-out/in, as well as group check-in.

A workspace can be started based on work-unit specification, and all components in the read-and write-sets can be checked out.

CHAT is a GUI to ECM and includes product and workspace browsers, as well as special support for negotiation on locked components as well as components which need integration before check-in or commit.

6.9 Cooperation Manager

Essentially interprets protocols according to specified events, and transfers data and messages accordingly.

Significant events and corresponding actions can be specified in work-unit descriptions or at run-time. The events are typically ECM operations (check-out/in, etc.), or message reception. The actions are ECM-commands and message sending.

These features are implemented by special daemons for each transaction. These daemons are integrated with CHAT, which provides message handling functionality.

6.10 PM Assistant (new)

Meta-process involves much more than Planner and Schema Manager, and more than SPELL: make a more general **PM Assistant**

- Intelligent assistant towards the human process modeler.
- Reuse well-proven and existing process models: must decide architectural formalism.
- Adapt, extend and evolve these; according to company rules and given feedbacks (metrics).
- Coupled to Experience Database.

Effort just started.

7 Experiences with EPOS model/tool architecture

7.1 Experiences with EPOS process models

Present modeling experiences:

1. Various toy examples.
2. A canonical demo example, both single and multi-user, 20 activities covering design and coding.
3. The ISPW7 example [CAJ⁺91] and parts of ISPW9.
4. Some process examples from Siemens and Ericsson.
5. Experiments with converting RCS based software archives to COV.

Ongoing:

1. Taking process/product models from nearby ESPRIT projects (REBOOT, PROTEUS), and from Sysdeco (sw house).
2. Using the EPOS code itself.
3. Taking aspects of enterprise models from Statoil.
4. An controlled experiment with NOVIT, a local bank software house. The study closely follows-up some representative projects to collect relevant metrics and data wrt. process evolution.

7.2 Experiences with EPOS tools

Current implementation of EPOSDB:

- Implemented in 30K lines of C, third rewritten version.
- Layered architecture, C-ISAM at bottom.
- Prolog API, built on internal C interface with SWI-Prolog.
- DB-mockup for single-user EPOS-PM.

Current implementation of EPOS-PM:

- About 10K lines of SWI-Prolog and XPCE.
- BMS add-on: easy.
- Prolog and XPCE: slow and buggy; but fast prototyping.
- EPOSDB + EPOS-PM in client-server mode: tricky debugging!

Also moving platform: Sun-2 to Sun-3 to Sun-4, SunOS to Solaris, ...

Much time used on (re-)implementation,
no direct industrial use and commercialization of this!

8 Conclusion

Preliminary model conclusions:

- One PML (SPELL) for product, activity, tool, and project submodels; and for meta-processes.
But other PMLs for versioning, transactions, ...
- But cryptic and low-level PML(s),
hard to see and understand total model.
Better viewing and UI needed!!
- Lacks a “soft path” to making activity models,
e.g. starting informally on instance-level.
- Remains work on: CM+PM, cooperating transactions,
QA/PrM and measurement-based process improvement.
- More high-level PML: general process architecture, ver-
sioning, ...
And associated method support!
- Meta-process is more than the SPELL domain!
- Modularize large and partly reusable process (sub)models.

Need more industrial experience!

Preliminary EPOSDB tool conclusions:

- Optimizations (also for COV).
- Add complex objects, consolidate transactions, ...
- Also API in C++?
- Commercial DBMS needed for Software Experience Database!
- Substitute EPOSDB with another (OO)DBMS?
cf. Marvel [BK92] and PMDB [PS91]:
OK for data model, but versioning / transactions?
- Implement cooperation/versioning model on e.g. groupware platforms?

Preliminary EPOS-PM tool conclusions:

- Good add-on to a (OO)DBMS.
Also via DB-mockup for testing.
- Better UI needed, in a general PM Assistant.
- More open process tool architecture, not one Prolog program.
- PML: now quasi-concurrent, fully OO, reflective Prolog!
Alternative basic PML: concurrent Prologs or Smalltalks, or CLOS, or commercial PMLs?
- Couple to tools for distributed systems and groupware, project management and QA?

Acknowledgements

Thanks go to colleagues in the EPOS project.

Selected Bibliography:

References

- [BEM93] Nouredine 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 Sandro Grigolli. Process Modeling In-the-Large with SLANG. In *[Ost93]*, pages 75–83, 1993.
- [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⁺92] Reidar Conradi et al. Design, Use, and Implementation of SPELL, A Language for Software Process Modeling and Evolution. In *[Der92]*, pages 167–177, 1992.
- [C⁺94] Reidar Conradi et al. EPOS: Object-Oriented and Cooperative Process Modelling. In *[FKN94]*, pages 33–70, 1994. Also as EPOS TR 198, NTH, 31 Oct. 1993, Trondheim.
- [CAJ⁺91] R. Conradi, A. Aarsten, M. L. Jaccheri, J. Larsen, C. Mazzi, N. M. Nguyen, and P. H. Westby. The EPOS Approach to the ISPW7 Software Process Modeling Example Problem. In *Proc. 7th International Software Process Workshop, 1991*, October 1991. 40 p.
- [CLH95] Reidar Conradi, Chunnian Liu, and Marianne Hagaseth. Planning Support for Cooperating Transactions in EPOS. *Information Systems*, 20(4):317–326, June 1995.
- [CM91] Reidar Conradi and Carl Chr. Malm. Cooperating Transactions and Workspaces in EPOS: Design and Preliminary Implementation. In Rudolf Andersen, Janis A. Bubenko jr., and Arne Sølvsberg, editors, *Proc. of CAiSE'91, the 3rd International Conference on Advanced Information Systems, Trondheim, Norway, 13–15 May 1991*, pages 375–392. LNCS 498, Springer Verlag, 578 p., 1991.
- [COWL91] Reidar Conradi, Espen Osjord, Per H. Westby, and Chunnian Liu. Initial Software Process Management in EPOS. *Software Engineering Journal (Special Issue on Software process and its support)*, 6(5):275–284, September 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.
- [Fel93] Stuart I. Feldman, editor. *Proceedings of the Fourth International Workshop on Software Configuration Management (SCM-4)*, Baltimore, Maryland, May 21–22, 1993.
- [Fer93] Christer Fernström. Process WEAVER: Adding Process Support to UNIX. In *[Ost93]*, pages 12–26, 1993.
- [FKN94] Anthony Finkelstein, Jeff Kramer, and Bashar A. Nuseibeh, editors. *Software Process Modelling and Technology*. Advanced Software Development Series, Research Studies Press/John Wiley & Sons, 1994. ISBN 0-86380-169-2, 362 p.

- [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).
- [LC93] Chunnian Liu and Reidar Conradi. Automatic Replanning of Task Networks for Process Model Evolution in EPOS. In *[SP93]*, pages 434–450, 1993.
- [Lie89] Anund Lie. *Outline Design of the EPOS Database*, April 1989. Draft, 38 p.
- [Liu91] Chunnian Liu. Software Process Planning and Execution: Coupling vs. Integration. In Rudolf Andersen, Janis A. Bubenko jr., and Arne Sølvberg, editors, *Proc. of CAiSE'91, the 3rd International Conference on Advanced Information Systems, Trondheim, Norway, 13–15 May 1991*, pages 356–374. LNCS 498, Springer Verlag, 578 p., 1991.
- [LL95] Jens-Otto Larsen and Patricia Lago. Transaction Technology for Process Modeling. In *[Wil95]*, pages 214–219, 1995.
- [MCL⁺95] Bjørn P. Munch, Reidar Conradi, Jens-Otto Larsen, Minh N. Nguyen, and Per H. Westby. Integrated Product and Process Management in EPOS. *Journal of Integrated CAE*, 1995. (Forthcoming in special issue on Integrated Product and Process Modeling), 30 p.
- [MLG⁺93] Bjørn P. Munch, Jens-Otto Larsen, Bjørn Gulla, Reidar Conradi, and Even-André Karlsson. Uniform Versioning: The Change-Oriented Model. In *[Fel93]*, pages 188–196, 1993.
- [NC93] Minh N. Nguyen and Reidar Conradi. Cooperating Transactions in a Versioned Database. In M. Huhns, M. P. Papazoglou, and G. Schlageter, editors, *Proceedings of ICICIS'93, International Conference on Intelligent and Cooperative Information Systems, May 12-14, Rotterdam, The Netherlands, May 1993*. IEEE Computer Society. Also as EPOS TR 165.
- [Ost93] Leon Osterweil, editor. *Proc. 2nd Int'l Conference on Software Process (ICSP'2), Berlin*. 170 p. IEEE-CS Press, March 1993.
- [PS91] Maria H. Penedo and Christine Shu. Acquiring Experiences with the modelling and implementation of the project life-cycle process: the PMDB work. *Software Engineering Journal (special issue on Software Process and its Support)*, 6(5):259–274, September 1991.
- [SP93] Ian Sommerville and Manfred Paul, editors. *Proc. 4th European Software Engineering Conference (Garmisch-Partenkirchen, FRG)*, Springer Verlag LNCS 717, September 1993.
- [Wil95] Wilhelm Schäfer, editor. *Proc. Fourth European Workshop on Software Process Technology (EWSPT'95), Noordwijkerhout, The Netherlands*. 261 p. Springer Verlag LNCS 913, April 1995.