(332)

(332)

mentation of the SCALE Computer-assisted Process Support Environment on PCTE", *Proceedings of the Third International Conference on PCTE (PCTE'94)*, San Francisco, USA, December 1994.

[76.] S. Achouba, I. Alloui, S. Latrous and F. Oquendo, "PEACE+: Support for Cooperation and Negotiation in Process-Centred Environments", *Software Process Improvement and Practice Journal*, (to appear).

[77.] I. Alloui, S. Arbaoui and F. Oquendo, "Process-Centered Environments: Support for Human-Environment Interaction and Environment-mediated Human Cooperation", *Proceedings of the 9th International Software Process Workshop,* Airlie, September 1994. IEEE Computer Society Press.

[78.] I. Alloui and F. Oquendo, “PEACE+ : une approche intentionnelle pour le support a la cooperation dans les environnements de genie logiciel assiste par ordinateur“, *Actes des 3èmes journées francophones IAD et SMA*, Chambery, March 1995. (in french)

[79.] I. Alloui and F. Oquendo, “PEACE+ : un formalisme et un systeme pour le support a la cooperation dans les environnements de genie logiciel centres processus”, *Proceedings of the 8th International Conference on Software Engineering and its Applications (GL'95)*, Paris, November 1995. (in french)

[80.] I. Alloui, S. Arbaoui, B. Debord, S. Latrous, F. Oquendo and G. Tassart, "SCALE/ALPS : un générateur de systèmes multi-agents pour la construction d'environnements centrés processus assistés par ordinateur", *Actes des 2èmes journées francophones IAD et SMA*, Voiron, May 1994. (in french)

[81.] J. Erceau and J. Ferber, Premières Journées Francophones IAD & SMA, Toulouse 1993. (in french)

[82.] S. Latrous and F. Oquendo, "PECAM: a ProcEss Centred Abstract Machine", *Research Report*, CRISS, Université Pierre Mendès France, Grenoble, 1994. (in french).

[83.] S. Latrous and F. Oquendo, “PEACE+/PECAM : une machine virtuelle pour l'exécution et l'évolution des processus logiciels”, *Proceedings of the 8th International Conference on Software Engineering and its Applications (GL'95),* Paris, November 1995. (in french)

[84.] F. Oquendo, “SCALE: A Next Generation Computer-Assisted Process Support Environment for System Integration”, *Proceedings of the 3rd IEEE International Conference on Systems Integration,* Sao Paulo, August 1994, IEEE Computer Society Press.

[85.] F. Oquendo, "SCALE: Process Modelling Formalism and Environment Framework for Goal-directed Cooperative Processes", *Proceedings of the 7th International Conference on Software Engineering Environments,* Noordwijkerhout, April 1995. IEEE Computer Society Press.

[86.]

(434)

(PhD thesis NTH 1993:78).

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

[61.] A. T. Nakagawa and K. Futatsugi. Software process á la algebra: Obj for obj. *Proc. 12th Int'l Conference on Software Engineering, Nice, France*, May 1990.

[62.] Bashar Nuseibeh, Jeff Kramer, and Anthony Finkelstein. Expressing the Relationship between Multiple Views in Requirements Specification. In *Proc. 15th Int'l Conference on Software Engineering, Baltimore, MA*, May 1993.

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

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

[65.] Mark C. Paulk, Bill Curtis, Mary Beth Chrissis, and Charles V. Weber. Capability Maturity Model for Software, Version 1.1. Technical Report CMU/SEI-93-TR-24, Software Engineering Institute, Pittsburgh, February 1993.

[66.] Dewayne E. Perry, editor. *Proc. 5th Int'l Software Process Workshop – ISPW'5*, Kennebunkport, Maine, USA, October 1989. IEEE Computer Society Press.

[67.] Maria H. Penedo and William Riddle, editors. *Materials from Process-sensitive Software Engineering Environment Architecture Workshop*, Denver, CO, USA, September 1992. Rocky Mountain Institute of Software Engineering, 1113 Spruce Street, Boulder, CO, 80302 USA.

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

[69.] Stanley M. Sutton et al. Programming a Software Requirements-Specification Process, in [64] .

[70.] Terje Totland and Reidar Conradi. A Survey and Classification of Some Research Areas Relevant to Software Process Modelling, 1995.

[71.] Kees M. van Hee and P. A. C. Verkoulen. Data, Process and Behaviour Modelling in an Integrated Specification Framework. In [66], pages 191–218, 1992.

[72.] T. E. White and L. Fischer (Eds.). *New Tools for New Times – The Workflow Paradigm*. Future Strategies Inc., Book Division, Alameda, CA, USA, 1994.

[73.] V.R. Basili,The Experience Factory and its Relationship to Other Improvement Paradigms,Proc.European Software Engineering Conference (ESEC'93),Garmisch, Germany, Springer Verlag LNCS 717, 1993.

[74.] J. Rumbaugh, et. al., Object-Oriented Modelling and Design. Englewood Cliffs, Prentice-Hall 1991.

[75.] S. Achouba, T. Agueda, P. Boulle, B. Debord and F. Oquendo, "Design and Implestructs for Managing Change in Process-Centered Environments. In *Proc. 4th ACM SIGSOFT Symposium on Software Development Environments, Irvine, California. In ACM SIGPLAN Notices, Dec. 1990*, pages 206–217, December 1990.

[46.] Yan Jin, Raymond E. Levitt, Tore Christiansen, and John C. Kunz. The “Virtual Design Team”: A Computational Model of Engineering Design Teams. In *AAAI’94 Spring Symposium on Computational Design*, March 1994. 7 p.

[47.] G. Junkermann, B. Peuschel, W. Schaefer, and S. Wolf. MERLIN: Supporting Cooperation in Software Development Through a Knowledge-Based Environment. In [31], pages 103–129, 1994.

[48.] R. Kadia. Issues Encountered in Building a Flexible Software Development Environment. In *Proc. of the 5th Symposium on Software Development Environments, SIGSOFT ‘92, pp. 169–180,* Reston, Virginia, December 1992.

[49.] T. Katayama. A Hierarchical and Functional Software Process Description and its Enaction. In *Proc. 11th Int’l Conference on Software Engineering, Pittsburgh, PA*, pages 343–352, 1989.

[50.] G.E.Kaiser, S.S. Popovich, and I.Z.Ben-Shaul. A Bi-Level Language for Software Process Modeling. In *Configuration Management, Ed. by Tichy, John Wiley and Sons Ltd 1994*.

[51.] 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.

[52.] Lung-Chun Liu and Ellis Horowitz. A formal model for software project management. *IEEE Transactions on Software Engineering*, 15(10):1280–1293, October 1989.

[53.] Dennis Lock. *Project Managment – Fifth Edition*. ISBN 0-566-07340-4. Gower Publishing, 1993.

[54.] C. Montangero and V. Ambriola. OIKOS: Constructing Process-Centred Environment. In [31], pages 131–152, 1994.

[55.] Nazim H. Madhavji. Environment evolution: The prism model of changes. *IEEE Trans. on Software Engineering*, SE-18(5):380–392, May 1992.

[56.] Ronni T. Marshak. *Workflow White Paper – An Overview of Workflow Software*, pages 15–41. International Workflow Coalition, 1993.

[57.] T. W. Malone, K. Crowston, J. Lee, and B. Pentland. Tools for Inventing Organizations – Toward a Handbook of Organizational Processes. Technical report, Center for Coordination Science, MIT, Boston, MA, USA, October 1992. 21 p.

[58.] N.H. Minsky. Law-Governed Systems. *Software Engineering Journal*, 6(5):285–302, September 1991.

[59.] Bjø rn P. Munch. *Versioning in a Software Engineering Database — the Change Oriented Way*. PhD thesis, DCST, NTH, Trondheim, Norway, August 1993. 265 p.

pages 12–26, 1993.

[31.] 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.

[32.] 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.

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

[34.] N. Goldman and K. Narayanaswamy. Software Evolution through Iterative Prototyping. In *Proc. 14th Int'l Conference on Software Engineering, Melbourne, Australia*, pages 158–172, May 1992.

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

[36.] M. Greenwood. Using CSP and System Dynamics as Process Engineering Tools. In *J.-C. Derniame (ed.): Proc. from EWSPT'92, Trondheim, Norway, Springer Verlag LNCS 635*, September 1992.

[37.] M. Hammer and J. Champy. *Reengineering the Corporation: A Manifesto for Business Revolution*. Nicholas Brealy Publishing, London, UK. 223 p., 1993.

[38.] Watts S. Humphrey and Marc I. Kellner. Software process Modelling: Principles of entity process models. In *Proc. 11th Int'l Conference on Software Engineering, Pittsburgh, PA*, 1989.

[39.] Karen E. Huff and V. R. Lesser. A Plan-Based Intelligent Assistant that supports the the Software Development Process. In *Proc. of the 3rd ACM Symposium on Software Development Environments*, pages 97–106, Boston, Massachusetts, November 1988.

[40.] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall International, 1985.

[41.] IEEE, editor. *Proc. of the 6th International Software Process Workshop – ISPW'6, IEEE Press*, Hakodate, Japan, October 1990.

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

[43.] ISO. *ISO9000-3:1991(E)International Standard*, 1991. (Quality management and quality assurance standards–Part 3: Guidelines for the application of ISO 9001 to the development supply and maintenance of software).

[44.] 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).

[45.] Stanley M. Sutton Jr., Dennis Heimbigner, and Leon Osterweil. Language Con-

[15.] I.Z. Ben-Shaul, G.E. Kaiser, and G.T. Heineman. An architecture for multi-user software development environments. *Computing Systems, The Journal of the USENIX Association, University of California Press*, 6(2):65–103, Spring 1993.

[16.] Reidar Conradi et al. EPOS: Object-Oriented and Cooperative Process Modelling. In [??], pages 33–70, 1994. Also as EPOS TR 198, NTH, 31 Oct. 1993, Trondheim.

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

[18.] G. Canals, N. Boudjlida, J. Derniame, C. Godart, and J. Lonchamp. ALF: A Framework for Building Process-Centred Software Engineering Environments. In [31], pages 153–185, 1994.

[19.] Reidar Conradi, Christer Fernström, and Alfonso Fuggetta. Concepts for Evolving Software Processes. In [??, pages 9–32, 1994. Also as EPOS TR 187, NTH, 9 Nov. 1992, 26 p., Trondheim.

[20.] Reidar Conradi, Christer Fernström, Alfonso Fuggetta, and Robert Snowdon. Towards a Reference Framework for Process Concepts. In [24], pages 3–17, 1992.

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

[22.] Bill Curtis, Mark I. Kellner, and J. Over. Process Modelling. *Comm. of the ACM*, 35(9):75–90, September 1992.

[23.] Reidar Conradi and Chunnian Liu. Process Modelling Languages: One or Many?, 1995.

[24.] T. H. Davenport. *Process Innovation – Reengineering Work through Information Technology*. Harvard Business School Press, Boston MA, USA, 1993. 337 p.

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

[26.] Wolfgang Deiters and Volker Gruhn. Managing Software Processes in the Environment MELMAC. In *Proc. 4th ACM SIGSOFT Symposium on Software Development Environments, Irvine, California. In ACM SIGPLAN Notices, Dec. 1990*, pages 193–205, December 1990.

[27.] 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.

[28.] G. Engels and L. Groenewegen. SOCCA: Specifications of Coordinated and Cooperative Activities. In [31], pages 71–102, 1994.

[29.] Clarence A. Ellis, S. J. Gibbs, and G. L. Rein. Groupware – Some Issues and Experiences. *Comm. of the ACM*, 34:39–58, January 1991.

[30.] Christer Fernström. Process WEAVER: Adding Process Support to UNIX. In [62],

Thanks go to colleagues in the PROMOTER project, and to the teams behind EPOS and E^3.

References

[1.] Vincenzo Ambriola, Reidar Conradi, and Alfonso Fuggetta. Experiences and Issues in Building and Using Process-centered Software Engineering Environments, December 1994. Internal draft paper V3.0, Univ. Pisa / NTH in Trondheim / Politecnico di Milano, 28 p.

[2.] Tarek Abdel-Hamid and Stuart E. Madnick. *Software Project Management - An Integrated Approach*. Prentice-Hall, 1990.

[3.] ANSI/NEMA. *Committee Draft 14258 – Industrial automation systems, Systems Architecture: Framework for Enterprise Modelling*. ISO TC184 SC5 WG1, June 1994. 32 p.

[4.] S. Arbaoui and F. Oquendo. PEACE: Goal-Oriented Logic-Based Formalism for Process Modelling. In [32] , pages 249–278, 1994.

[5.] R. F. Bruynooghe et al. PADM: Towards a Total Process Modelling System. In [31], pages 293–334, 1994.

[6.] Noureddine Belkhatir, Jacky Estublier, and Walcelio Melo. Software Process Model and Work Space Control in the Adele System. In [62], pages 2–11, 1993.

[7.] N. Belkhatir, J. Estublier, and W. Melo. ADELE-TEMPO: An Environment to Support Process Modelling and Enaction. In [31], pages 187–222, 1994.

[8.] 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).

[9.] S. Bandinelli, A. Fuggetta, C. Ghezzi, and L. Lavazza. SPADE: An Environment for Software Process Analysis, Design, and Enactment. In [31], pages 223–247, 1994.

[10.] M. Baldi, S. Gai, M.L. Jaccheri, and P. Lago. Object Oriented Software Process Model Design in E^3. In [31], pages 279–290, 1994.

[11.] 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.

[12.] Naser S. Barghouti and David S. Rosenblum. A Case Study in Modeling a Human-Intensive, Corporate Software Process. In (see [64]).

[13.] Victor R. Basili and H. Dieter Rombach. The TAME project: Towards improvement-oriented software environments. *IEEE Trans. on Software Engineering*, SE-14(6):758–773, June 1988.

[14.] I. Z. Ben-Shaul and Gail E. Kaiser. A Paradigm for Decentralized Process Modelling and its Realization in the Oz Environment. In *Proc. 16th Int'l Conference on Software Engineering, Sorrento, Italia*, pages 179–190, May 1994.

digms for the core PML, in order to find *the* correct one. Our position is, that the idea of one, standard, all-encompassing PML is utopia, because of theoretical problems and interoperability requirements. And we should lift the perspective up from low-level activity formalisms.

To recapitulate: there is a big variety of process phases and elements to be covered, although the community has concentrated much on the Implementation 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/L5** 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 primary process elements, is not so important. What really counts is:

•**Actively reusing technology** from related domains:
Information Systems, Enterprise Modelling, Business Process Reengineering, Project Management, Quality Assurance – conceptually and technically.

•**Standardization**: common PM **concepts / formalisms?**
At least a taxonomy for classification and assessment of such, and of common process models – cf. ISPW examples is needed.

More concretely: Reuse of standard support technologies, e.g., Unix/MS-DOS, C++, CORBA or ODMG, X/Motif, is a keyword.

•**Interoperability**:
Making PSEE components interact smoothly with other process and production models and tools. Modularization, open systems and above standardization are keywords here. Some concrete issues to consider are:
– coupling to CM: multi-actor and distributed.
– coupling to Project Management.
– coupling to Quality Assurance (ISO 9001, CMM).
– coupling to Groupware and social processes.
– coupling to Organisational theory (BPR, EM).

(433) From a PMl perspective, this means that the different languages, offered by the existing systems have to be integrated around

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

Acknowledgments

PML ------ MP Phases	APPL/A	Marvel	Process Weaver
Enac- tion	APPL/A	opportu- nistic process- ing	PML + BMS
Assess- ment			

language?

(431)

3.6 Architectural PSEE Implications of PMLs

(432) This section discusses general architectural issues: how the PML is going to be implemented by a PSEE. A general software architecture for a PML comprises at least:

1. a repository in which the model is stored and maintained in a versioned fashion. A general PSEE architecture is centered around 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. In most PSEEs, all the (sub)models are stored in a common repository.
2. an editor to facilitates model creation, and manipulation. A semi-formal PML must be given with a formal syntax and can be supported by automatic editing (graphical) tools.A semi-formal PML must be given with a formal syntax and can be supported by automatic editing (graphical) tools.A semi-formal PML must be given with a formal syntax and can be supported by automatic editing (graphical) tools.
3. an intepreter to execute models. Formal semantics is usually given in an operational way by providing an interpreter, or compiler, for the language. In this case, process models are said to be enactable. Model interpretation, i.e., enaction, involves external tool interaction. Here, it can be strategic to consider standard interfaces, e.g., those defined by CORBA or ODMG. As for any interactive system, a user interface is needed by which a user of the PSEE can interact with the process engine. Furthermore, the current state of the process, i.e. a view on the instances of the process model, may be shown to the user during a running software process. Parts of the process modelcan be 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.

3.7 Conclusion

Until recently, much research in process modelling has been focussed on different linguistic para-

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

PMLs ------ Process Elements	APPL/A (L2)	MARVEL (L3, L5)	Process Weaver (L5)
Product	variant records and relations	class	No, just place-holders
Role		class	parametric work contexts
Human			instantiated work contexts
Tool		tool envelope	tool class
Evolution		rules	No
Projects	predicates		cooperative procedures
Work-context			
Tool view			Unix tools
User-View			agenda
Cooperation			coopra-tive procedures
Versioning			
Quality			

(430)

Table 6: Meta-process phase and existing PMLs: which phase is covered by which sub-

PML ------ MP Phases	APPL/A	Marvel	Process Weaver
Elicitation		ASL	method level
Anal ysis			
Design		ASL	Cooperative procedures
Implementation	APPL/A	MSL	Co-Shell

```
        teams : set_of link TEAM;
        current_tasks : set_of link TASK;
end
```

(424) Rules declare pre- and post-conditions and a code part. In the following, we give a very simple example of a rule [12] .

```
solve_MR[?mr:MR] :
        :
        no_chain(?mr.status = open) #precondition
        {IMRDB mrsolve ?mr.name}
        (?mr.status = solved);
```

(425) The rule declaration models the creation of a code solution for an individual modification request, after which the status of the modification request changes from open to solved.

(426) Recently, the Activity Structures Language (ASL) [50] has been implemented in the context of Marvel. ASL is a variant of Riddle's constraint expressions that enables to model process model topology by means of regular expressions extended with concurrency operators. ASL fragments can be translated into MSL ones.

3.5.3 Process Weaver

(427) Process Weaver is based on an internal PML, hidden to the user, and manipulatable by a number of views. The design paradigm is thus L5. The elicitation meta-phase is facilitated by the highest abstraction level called the method level. Here, we can define decomposition hierarchies of activity types. An activity type can be regarded as a specification, or interface, that can be associated with a textual description of related roles, products, and techniques. Process models can be designed by cooperative procedures that express activity control flow. Cooperative procedures are expressed in a Petri net like language. Each Petri net transition can be associated with a pre-condition and an action that are expressed in the CoShell PML. Work assignments can be described by a special PML with so-called Work Contexts. The WorkContext level offers tool classes to map from Work Contexts to local tools. Product modelling is not explicitly supported and product place holders can be expressed by CoShell variables.

(428) Process Weaver provides special graphic editors to manipulate activity types, cooperative procedures, CoShell conditions and actions, and Work Contexts.

(429)

Table 5: Process elements and existing PMLs

PMLs ------ Process Elements	APPL/A (L2)	MARVEL (L3, L5)	Process Weaver (L5)
Activity	Ada tasks and procedures triggers	class rule	Activity type (interface) transition (body)

3.5 Other PMLs

(418) Here, we provide a survey of three PMLs developed outside Promoter. They are the APPL/A PML used in ARCADIA and the three MARVEL PMLs, both pioneering research PSEEs and the Process Weaver PML from a pioneering commercial system. Also, these three PMLs are representative for three main linguistic paradigms that are respectively procedural (APPL/A), rule based (Marvel), and Petri net based (Process Weaver).

3.5.1 APPL/A

(419) APPL/A has been developed in the context of the ARCADIA project and it is a process programming language. APPL/A is a superset of Ada, extended with persistency, relation management, transactions, and triggers. The language is thus mainly procedural, but consistency constraints can be expressed as predicates on relationships. In the following we give an excerpt of a process program coded in APPL/A and taken from [69] :

(420)

```
Procedure SingleUserModel is ...
Begin
    while not Done loop
        case CurrentApproach is
            when TopDown =>
                if NeedRool then CreateRoot;
                elsif NeedChildren then
                    SelectNodeToElaborate(currentNode);CreateChildren(CurrentNode);
                else SelectNodeToEdit(CurrentNode);EditNode(CurrentNode);
                end if;
...
```

3.5.2 MARVEL

(421) Several published papers describe the evolution of the Marvel environment. Marvel 3.1 offers a process modeling language called MSL, with three main constructs being classes, rules, and tool envelopes.

(422) Classes have attributes that can be typed attributes or links to other classes. The following examples, taken from [12] show some simple class definitions:

(423)

```
SUBSYSTEM :: superclass ENTITY;
    MRs : set_of MR;
    modules : set_of MODULE;
end

MR: superclass ENTITY; #modification request
    contents: text;
    status : (new, open, screened, inprog) = new;
    name: text;
end

EMPLOYEE :: superclass ENTiTY;
    level : (MTS, TechManager, Dept, Head, Secy, Office_manager) = MTS
    # default value is MTS
    login : user; #the login id
    office : string;
    phone : string;
    dept : string; #department number
    boss : link EMPLOYEE;
```

PMLs ------ Process Element	Spell (L2, L4)	Socca (L5)	Merlin (L3)	Oikos (L3)	Alf (L3)	Adele (L3)	Slang (L3,L4)	Peace+ (L3)	E3 (L5)	Padm (L2,L3)
Quality								No	IV	

(417) Table 4 gives, for each PML, an indication of how, and if, each meta-process phase is supported.

Table 4: Meta-process phase and existing PMLs: which phase is covered by which sub-

PML ------ MP Phases	Spell	Socca	Merlin	Oikos	Alf	Adele	Slang	Peace+	E3	Padm
Elicita-tion		STD + Para-digm + Class Dia-grams	Escape	Limbo	ALF PML	APEL	SLANG	PDD	E3 PML	BM
Anal ysis		Simula-tion: STD + Para-digm		Limbo	ALF PML	APEL	SLANG	PDD (partly)	E3 PML	BM
Design		STD + Para-digm + Class Dia-grams	ESCAP E	Limbo	ALF PML	APEL	SLANG	PDD	E3 PML	BM
Imple-menta-tion	SPELL	STD + Para-digm + Class Dia-grams	Mer-linPML	Pate’	ALF PML	Adele PML	SLANG	PML	E3 PML	PML + PWI
Enac-tion	SPELL + EPOS-DB + Unix-tools	No	Merlin PML	Pate’	ALF PML	Adele PML	SLANG + O2 + DecFuse	PML + PCTE	NO	PML + PWI
Assess-ment		STD + Para-digm + Class Dia-grams						No		BM + PML + PWI

language?

3.4.12 Discussion

(416) In Table 3, we try to give a taxonomy of how (and if) the different PMLs represent the different process elements that were introduced before. We also attempt to classify each PML according to its design approach. Note that some PMLs have been classified under more than one design approaches.

Table 3: Process elements and existing PMLs

PMLs ------ Process Element	Spell (L2, L4)	Socca (L5)	Merlin (L3)	Oikos (L3)	Alf (L3)	Adele (L3)	Slang (L3,L4)	Peace+ (L3)	E3 (L5)	Padm (L2,L3)
Activity	(L2) task type	STD	process level (facts)	Limbo Pate'	rules costraints character-istics	Apel Tempo Adele	transi-tion activity type	PML task concept	IV TV TDV	PML
Product	(L2) data type	Class Dia-grams	process level (facts)	Limbo Pate'	ERA		OO part	PCTE (OMS/ DDL)	IV TV UV	PML
Role	(L2) role type	Internal Behav-iour (STD)						PML role concept	IV TV UV	PML
Human		Class Diagram						PCTE		PML
Tool	(L2) tool type	Class Diagram		Limbo Pate'	operator type		black transi-tion	PML agent opera-tor con-cept	IV TV UV	PML
Evolu-tion	(L2) type level methods	Woodan Ana-crhronis tic pro-cesses					activity type	PML		PML
Projects	(L4) EPOS-DB	SOCCA model	project level (facts)					PML	UV	
Work-context	(L4) EPOS-DB	Current States of Inter-nal STDs						PCTE		
Tool view	(L4) UNIX Tools						DEC FUSE SUN TOOL TALK (L4)	PCTE	IV TV UV	PML
User-View								User inter-face agents	UV	PML
Cooper-ation	(L4) EPOS-DB	Para-digm	Rules					PML agent interac-tion models		
Version-ing	(L4) EPOS-DB						OO DB O2 (L4)	PCTE (VCM)		

Attributes and methods can be defined for classes and associations (implementation meta-phase).

(409) For the E3 PML, there is a drawing tool that enables to define, browse, and reuse process model fragments according to four views that implement the inheritance view(IV), task view (TV), functional decomposition (TDV), and informational perspective(UV). The tool implements syntactic and static semantics checks on process models thus facilitating the analysis meta-phase. The E3 PML has been designed with the only goal of providing an easy support for process model definition.

3.4.11 PADM

(410) The Process Analysis and Design Methodology (PADM) is an example of moving from a L2 to L3 design paradigm. Its exemplar enactment system is the ICL product ProcessWise Integrator (PWI), formerly IPSE 2.5, which coordinates users and tools by enacting a process written in its PML. PWI PML is class based, providing base classes and reflexive features covering the primary process elements: Action, Entity, Role, Interaction, User Agent, Tool Agent, GetClassDefinition and BehaveAs. The ability to reuse specializations of these classes places PWI PML in the L2 paradigm. It covers all the meta-process phases though most emphasis is on enactment and evolution.

(411) Experience with PWI PML showed that the detail needed for enactment made it difficult to use for model elicitation or investigating design alternatives. An effective enactment language needed at least one design method to exploit it.

(412) Base Model (BM) provides a specification language which supports the gradual refinement of a detailed design from a high-level specification. Its underlying temporal logic semantics can be exploited not only to prove properties of models, but also to prove the correctness of the refinements. BM therefore concentrates on the specification, analysis and design meta-process stages. BM does not deal directly with process elements but with the abstract concepts of objects and operations. This enables the modeller to focus on one aspect of the model during the initial development staged and then add further details.

(413) The combination of BM and PWI PML fits the L3 paradigm. Ideally it would be possible to formally refine from the most detailed BM model into PWI PML, but currently some hand translation is needed. The BM stepper can be used to assist checking the consistency of the BM and PML. The BM-PWI PML combination means that each language can exploit its strengths. BM provides a formal development process. PWI PML deals with the enactment details such as the interface between enactment engine and tools, and the incremental change of process instances.

(414) BM is influenced by its underlying temporal logic semantics, and formal specification approaches in general. PWI PML is influenced by object-oriented languages, reflexive systems, and role-interaction theory.

(415) Within PADM there has also been work on Conceptual Models, and alternative to BM for modellers who prefer informality in the early stages. In addition, the latest releases of PWI also offer separate facilities to program the model-tool and model-user interfaces. All of these confirm the general L2 to L3 trend.

reflection and reification [58] [83] . PEACE+/PML is an enactable knowledge representation modelling formalism based on the multi-agent paradigm of Distributed Artificial Intelligence field [36] . The cooperation aspects rely on an intentional approach of rational action [78] [79] . A software process is seen as a multi-agent system where agents that are cognitive, rational and intentional assist users in enacting process model tasks.

(402) PEACE+/PML consists of two compatible sub-languages (L3): The data definition language of the PCTE/OMS to represent the object model and the PME/DL (process model element description language) to represent the activity model. Process models elements are defined in terms of user performer roles, process model tasks and process model agents.

(403) User performer roles define the roles members of a project may play in terms of its view on the process object base, its privileges, its obligations and the required qualifications for a performer to be connected to the role (position, skill, experience, etc.).

(404) A process model task is defined by input/output typed parameters (expressed in the PCTE/OMS DDL), a precondition i.e. a necessary condition for starting its enactment and finally, the goal that should be satisfied by the executing agents of the task, the role a performer must have to enact the task and the kind of interaction with this latter (interactive, non interactive).

(405) A process model agent is defined by its knowledge on objects and tasks of the software process, its capabilities in terms of operators on process tasks and objects, its interaction capabilities in terms of acquaintances and interaction models, and its goal. A process model agent acquaintances define the other agents it knows about and with which it can interact (communicate, coordinate and cooperate) by using interaction models. Each interaction model is defined by a set of states and a set of transitions. A transition is done through the realisation of a communication act which may be a request, a notification, a command, etc.).

(406) Knowledge, preconditions, postconditions and goals are expressed using basic operators of belief, intention, necessity and possiblity of a multi-modal logic. This constitutes a good trial in applying formal techniques to process Modelling. The advantage of using multi-modal logic is that it allows non-monotonic reasoning which helps to support process dynamicity and evolvability that result from incomplete and uncertain knowledge during the process enactment [4] . PEACE+/PML as a reflective language, allows users to define, enact and evolve software processes as well as meta-processes in a uniform way.

(407) In the framework of the SCALE ESPRIT Project, it also has been developed a graphic language called Process Description Diagram (PDD), designed for process understanding and design, and from which PML definitions can be generated [73] [85] .

3.4.10 E3

(408) E3 offers an object oriented analysis and design notation that is suitable for the meta-process high phases and is not enactable. E3 PML offers pre-defined classes and associations which are given with intuitive graphic symbols. User defined classes (respectively associations) inherit graphic symbols from their super classes (respectively associations).

trigger execution is difficult to control and monitor. To overcome the limitations of the first PML, the process formalism Tempo has been designed. TEMPO is based on the notions of role (defining both static and dynamic object properties) and connection (expressing how objects collaborate).

(394) Tempo was designed to offer support for Modelling of primary process elements. Further, special emphasis is given to cooperation, versioning, and configuration management Modelling.

(395) Adele has its roots in object-oriented languages, database Modelling, and trigger mechanisms.

(396) In the framework of the Perfect Esprit project, APEL (Abstract Process Engine Language) language was developed. APEL is a graphical high level language, designed both for process capture and understanding and for its enaction. It can be compiled toward an abstract Process Engine built from two commercial basic process engines which interoperate on a peer to peer basis: Adele and Process Weaver. In APEL a process is represented using the 5 different aspects: Control flow, data flow, data description (OMT like), Work Space and cooperation, (user) roles, and state transition diagram.

(397) Aspects are interrelated and consistency is enforced, but there is no main aspect. Users can describe the process using all the aspects, or a subset of them. Each description is an abstraction level. Each level can be recursively refined.

3.4.8 SPADE

(398) SPADE PML is called SLANG. SLANG consists of two layered languages: kernel SLANG that is a formal language based on Petri nets (for behavior definition) and object orientation (for static structure definition), and full SLANG that enriches kernel SLANG with PM specific syntactic sugar and with special types for process Modelling evolution. SLANG is integrated with the O2 object oriented data base, which acts as a repository for both process model and process products.

(399) In addition, it is possible to model cooperation policies as part of the process model. This is achieved through process-tool interaction mechanisms embedded in special language constructs (black transitions and user places). Tools can be integrated directly in the SPADE environment or via commercial tool integration products, such as DEC FUSE and SUN ToolTalk.

(400) The design paradigm is L3. SLANG should be suitable for all meta-processes phases as Petri nets are an analysis and simulation tool and the language is enactable. SLANG can be used to model and enact the meta-process as well.

3.4.9 PEACE+

(401) PEACE+ is the second phase of PEACE [4] . It enhances PEACE with (1) cooperation formalism and support that allow to describe and enact interaction models (communication, coordination, negotiation, etc.) during enaction of distributed process models [80] [77] [76] , (2) a process models enactment and evolution system based on structural and operational

an executable distributed language. The meta-process is based on step-wise refinement of LIMBO specification into Pate` executable code.

(386) Both languages are concurrent logic ones derived from Extended Shared Prolog (EXP). The distinguishing features of an OIKOS process model are the blackboard pattern of communication among concurrent agents. Activities are arranged in a hierarchy that provides the main abstraction/modularization axis.

(387) No specific languages support for tool and data description is provided. Tools and data reside in external repositories and are specified by a signature. Design paradigm is L3.

(388) A graphical notation for Limbo is supported by an editor and browser. Analysis is supported only insofar as it is possible to animate a specification, i.e. run it in a simulated environment, without actual tools and documents.

3.4.6 ALF

(389) A generic process model, in ALF, consists of fragments, called MASP (Model for Assisted Software Process). The ALF PML comprises an Extended Entity Relationship Attribute (ERA) data model to describe the data part; an operator type sub-language for tool description; a rule sub-language to express in an event-condition-action fashion, how certain events have to be processed; a constraint sub-language to define, by means of path expressions, operator invocation order; a first order logic language to define "characteristics" that is an expression which has to be true and is used as invariant or objective.

(390) The language consists of five compatible sub-languages (L3). The logical re-formulation of the extended entity relationship data model is the core language. MASPs can be arranged in hierarchies where an operator type is itself a MASP. Import/Export mechanisms between MASPs are provided. Static evolution is facilitated by the MASP modularization facilities, dynamic evolution can be done either during instantiation (by parametrization) or after, during execution, by instance modification.

(391) A generic model can be instantiated into a project specific enactable model, that is called IMASP. Instantiation is either statically or dynamically. ALF PML is not meta-process phase specific, but it should be suitable for incremental development, from specification to implementation, due to its good specialization mechanisms. The language is given with operational semantics. Some rules and mechanisms are provided to detect inconsistencies and possible inconsistencies in a MASP specification (Inconsistency Tracker For MASPs).

(392) ALF has its roots in ER data Modelling, active databases, and logic programming.

3.4.7 ADELE-TEMPO

(393) ADELE offers two PMLs: historically, the first language, is an object oriented database language extended with relationships and triggers. Documents, tools, and work contexts are the primary elements of this enactable language. Activities can be expressed by means of triggers, that are condition/action rule associated to data base operations. The Adele first language has been evaluated to be good for process implementation and enaction, but not for specification and assessment as the formalism is difficult for human to understand and

Different views, i.e., static and behavioral, are offered. Abstraction and modularization derives from object-oriented principles such as classification and aggregation, however these have not been tailored to process Modelling. No specific support is devoted to customization and evolution.

(380) SOCCA has been clearly influenced by information system Modelling. SOCCA is related to OMT which it extends by explicitly addressing communication.

3.4.4 Merlin

(381) In Merlin, process modeling is performed on two different levels. The first level is visible for the process-engineer and represented by the process-design language ESCAPE. The second level is used for enacting the process design and represented by the Prolog-like process programming language.

(382) The process-design language ESCAPE (Extended Entity Relationship Models and Statecharts Combined for Advanced Process Engineering) is a graphical language which distinguishes between three different models: (1) the object-model, (2) the coordination-model and (3) the organization-model. The object-model is based on the EER-model and used to specify the structural aspects of the process, e.g. the document-types, the activities, the relationships. The coordination-model is based on Statecharts and used to specify the behavioural aspects of the process, e.g. the pre- and postconditions for activities, the change conditions for state changes. The organizational-model is based on tables and used to specify the organizatiobnal aspects of a process, i.e. the roles and responsibilities.

(383) ESCAPE designs are understandable because the concepts used are well known from information system modeling (e.g. OMT), the structuring of processes is reached through separation of concern basic concepts like for example transaction management can be presumed and must not be part of the process model itself. A further advantage of ESCAPE is, that an application specific analysis is supported. The process-design is checked for errors, warnings and optimizations. By that, the quality of processes can be improved before enacting them.

(384) The enaction of a process is reached by automatically mapping the ESCAPE design to the enactable Prolog-like process programming language. The ESCAPE-definition is mapped to PROLOG-facts. PROLOG-rules specify the semantics, i.e.how the before introduced facts are interpreted. This includes to define how the information to build the working context is selected, how transaction management is supported, how version management is handled etc.. Process- and project level are variant and change for every process specified while th rules which are also called cooperation-model, are the invariant part of the process programm. Currently, no support for simulation (dry runs) is given. ESCAPE and thus the process modeling language has its roots in information system modeling.

3.4.5 OIKOS

(385) OIKOS offers two PMLs, named LIMBO and Pate'. LIMBO is the specification language to be used during high level meta-process phases such as specification and design. Pate' is

3.4.2 EPOS SPELL

(373) EPOS SPELL that is the EPOS PML, consists of an extensible kernel (L2) of pre-defined entity and relation types (task, data, tool, and role) that can be specialized by inheritance. The Task type declares PRE and POST-conditions that are first order logic expressions (as in AI); FORMAL that are input output parameters declaration (as in data flow diagrams); DECOMPOSITION that defines task breakdown (as in functional decomposition). SPELL is implemented on top of an object oriented database, that provides support for persistency, versioning, and transactions (L4). It is a reflective language that provides support for both evolution and meta-process explicit definition.

(374) SPELL types can be defined by a textual Prolog based notation that is suitable for implementation, but not for the meta-process higher level phases. SPELL instances can be visualized in a graphic environment that gives support for process assessment.

(375) EPOS has constituted one of the first attempts in merging different paradigms. SPELL finds its roots in object oriented languages and databases, reflective systems, AI planning, and data flow systems.

3.4.3 SOCCA

(376) SOCCA is a specification language that offers a class diagram for the data perspective, state transition diagrams for the behavior perspective, PARADIGM for the communication, and that will offer an object flow diagram for the input output or process perspective. The design paradigm can be classified as L5, as there is no core language, rather several PMLs.

(377) The main purposes of modelling are declared to be analysis and expressing for humans. The PML in its current version is mainly concentrating on the first meta-process phases, i.e. requirement specification, analysis and design, and on assessment. Iteration of these phases also is allowed, e.g. redesign after assessment, particularly in view of change. For the future more support for implementation and enaction are envisaged. Support for simulation is given.

(378) Each process element in a process model, appears in a class diagram and is associated with attributes and methods. Process elements, i.e. classes, are connected to each other by relations. For each class, the interface (external behavior) and implementation (internal behavior) are specified by state transition diagrams. Consistency between the classes and the STDs is assured by labelling transitions from external STDs with methods from the corresponding class. Communication between the STDs is modelled through PARADIGM. Further integration of the data and behavior perspectives is achieved by describing the effect of a transition in terms of insertion, update, and deletion of class and relations instances. No specific language support is given to the Modelling of different process artifacts, i.e., even the primary process elements (e.g., activity, artifacts, humans and roles, tools, and support for meta-process) have not been given with special syntactic and semantics support.

(379) SOCCA diagrammatic notations help in understanding process models. SOCCA is simulatable, even if not truly enactable, language. It is the STD sub-PML, in combination with PARADIGM, which guides interpretation. Support for analyzability relies in simulation.

are covered by existing or standardized “PMLs” and associated tools, but mutually possibly not homogeneous. We can mention project models, data descriptions used by CASE-tools, tool configuration schemes in broadcast message servers, and issues related to user interfaces or configuration management. Thus, **PML interoperability** and standardization becomes a crucial issue, cf. **federated** databases and **standardization** work in International Workflow Management Coalition [Mars93] and OMG [Grou92].

That is, what is the **core PML** and what are the **non-core PMLs**, and how are all these related? 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 labor and risk, and enlarge as insight and confidence is gained – cf. product Modelling by minimal placeholders in Process Weaver.

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 the following, we are going to survey some PMLs. Section 3.4 is about PMLs developed by those participating to promoter project and Section 3.5 is about three other PMLs.

3.4 Process Modelling Languages in the context of Promoter project

3.4.1 The survey method

(368) The purposes of this survey are to both provide examples for the PML issues identified previously in this chapter and summarize the main features of the existing PMLs. The process of producing this survey consisted of the following steps: 1) study of background material; 2) production of a PML classification; 3) distribution of this classification to PML design ers;4) PML designers provide feedback; 5) subsequent adaptation of the PML classification.

(369) To make this process feasible, it has been decided to provide a classification of the European PMLs in the context of the Promoter project. The reasons for this choice are:

(370) 1) there exist a uniform background material that has been already been classified and assessed by Lonchamp work in the first PM book; 2) PML designers can be easily communicated by the Promoter net. Not European PMLs will be not neglected, as an appendix is dedicated to them.

(371) The grid for this classification is derived from the issues discussed previously in this chapter. For each PML, we will try to answer the following questions: a) which is the PML design paradigm (L1-L5)? b) which meta-process phases is the PML devoted to? c) which process elements can be described? d) which are the technical requirements that are better fulfilled by the PML? e) which domains or technologies the PML has been influenced by?

(372) The order the PMLs will be presented is the same as in the first PM book. Table 3 will summarize PML features by assessing which process element is modeled and if yes by which sub-language. Table 4 on page 75 summarizes the discussion about PML features and metaphase support.

PML can be formulated. Thus, fundamentally different PMLs and PML notations may be needed to cover such diversity in scope, coverage, abstraction, granularity, customizing, and user views [Conra94e][Ambr94].

PMLs that reflect different linguistic paradigms, e.g. rule-based [Shau93], Petrinets [Band94] , and imperative [Kadi92], have been presented in literature. No PML has been demonstrated to satisfy all the PM requirements, and a real assessment of the existing PMLs does not exist.

We can identify four approaches named **L1–L5** for PML design, and all include a **core** PML.

- **L1: One fixed and large core PML**. Here, we have one core large PML that contains language primitives to express all relevant process elements. E.g. **Activity, evolution support**.

  Typical examples of “hard” language primitives include constructs for concurrent activities and reflexivity.
- **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.

  E.g. **Product, role, human, tool**. “Soft” primitives, such as extensible types, or descriptions of product structures.
- **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.
- **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. Cooperation, versioning, transactions, and quality (metrics) performance issues seem candidate to be expressed by a separate incompatible PML.
- **L5**: One large language, plus views. This is similar to L3 above, the difference relies in the fact that a view is a part of a language, not a real language.

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. [Hee92]), 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, schemes and instances from possibly heterogeneous subdatabases.

However, sometimes we do not have a free design choice, since parts of the PM domain *already*

3.3.6 CASE Tools and tool integration mechanisms

(366) Some PMLs delegate an external engine for tool description and integration, e.g., SPADE uses DecFuse.

3.3.7 WorkFlow and Groupware

(367) Role-Interaction-Diagrams and groupware, e.g., Action Workflow.

Table 2: Other domains and process element coverage.

Legend H= hinders, A= assists, N= needed

Other domains ------ Process Elements	Project Man.	Formal Spec. Languages	Semi-formal Design Notations	Progr Languages	Metric-centred	Database	CASE Tools	Workflow and Groupware
Activity (P)	A	A concurrent	A: behavioral statical		A			A
Product (P)		A not concurrent	A statical functional		A	N		
Role (P)	A		A					A
Human (P)	A							A
Tool (P)							A	
Evolution (P)						N		
Projects (S)	A				A			A
Workcontext (S)						A	A	
Tool view (S)						A	A	
User View (S)	N							
Cooperation (A)						N		A
Versioning (A)						N		
Quality (A)					A	A		

3.3.8 The PML Design Dilemma: One or many PMLs?

The search of *the ideal PML* has constituted one of the main PM efforts in the last years. The process Modelling community has expressed tough requirements on such a PML: It must comprehensibly and understandably express the **primary process elements**, e.g. activities, products etc. (Section 3.2.1). It must enable expression of both template, enactable and enacting process models, where the two former may be either company-generic or project-specific. It must offer facilities for structuring, modularization, customizing, and evolution of such [Perr89][IEEE94]

Depending on the purpose of PM, e.g. to provide human understanding vs. static analysis vs. dynamic enactment of process models, different and partly conflicting demands on the underlying sons and activities. Activity networks show activity inter-dependencies. In [Liu89] activity network can be generated by PML descriptions. Process Weaver has been coupled to a commercial project management tool, e.g. MicroSoft/Project. This allows use of common project management techniques to perform critical path analysis etc. Other PSEEs that are characterized by the project management approach are CADES, ISTAR, and the Virtual Design Team.

3.3.2 Formal Specification Languages

(360) A formal specification language is a language whose syntax, and semantics are formally defined. A formal specification is a mathematical object that can be analyzed using mathematical methods. Many PMLs are based on Petri net. In [Gree92] the specification language CSP [Hoar85] is investigated as a basis for a PML and in [Naka90] the specification language OBJ is exploited. Specification languages fall into many categories: model oriented or property oriented, concurrent or not, modular or not. In Table 2, we will distinguish among these three types of languages, when evaluating PML technologies against process element coverage.

(361) We believe that PML research must exploit as much as possible the result that come from the specification language community as mathematical methods for PM are still lacking.

3.3.3 Informal Design Notations

(362) Design notations, e.g., object oriented design notations, are usually less formally defined than specification languages. Statemate has been successfully exploited as the basis for EPM [Hump89]. In E^3, an object oriented design notation has been taken as the basis for the PML. Here we distinguish between three kinds of notations: devoted to static descriptions, e.g., ER; behavioral description, e.g., state transition diagrams; devoted to functional description, e.g., data flow diagrams.

3.3.4 Programming languages

(363) As a first reaction of the consideration “Software processes are software too”, existing programming languages, e.g. ADA, have been exploited as PML kernels. Even if they suffer the granularity problems, e.g., language first order objects are low level ones, such as integers and real, they have been crucial for first PM experiments.

(364) Prolog has been used as a prototyping vehicle for the majority of the PMLs above. Also, the object oriented paradigm lies at the bottom of most of the PMLs.

3.3.5 Data base languages

(365) ADELE exploits a data base description language to express process models. Almost each PML is integrated with an underlying database to store products. Active databases may assist in representing tool effects, e.g., triggers.

abstract submodels which are customized within a concrete process model. Also, a PML may or may not offer the possibility to distinguish among generic (template) and specific (instantiated) process models.

(354) • **Executability**: PML may support to define **operational** models. Operational models are executable and easily **enactable**.

(355) • **Analyzability**: PML may support to define **descriptive** models, e.g., predicate logic expressions. Descriptive models are easily **analyzable**.

(356) • PML may directly support the evolution of PMs. **Reflection** is one important requirement. Then, there are parameterization, dynamic binding, persistency and versioning.

(357) • **Multiple conceptual perspectives/views**: PML may support the definition of (consistent) **views** on certain perspectives of a process model. This implies mechanisms to **integrate** different views on a process model into a common, overall process model.

(358) Table 1 shows, for each meta-process phase, which PML characteristics are either needed,

Table 1: Requirements for meta-process phases

Legend H= hinders, A= assists, N= needed, blank = neutral

Characteristics/ Phase	Formality	Graphic	Abstraction/ Modularity	Executability	Analyzability	Evolution support	Multiple views
Elicitation	H	A	A	H			A
Analysis	N				N		
Design	A	A	N		A		A
Implementation	N		A		A		A
Enaction				N	A		A
Assessment		A					A

or assisting, or hindering.

3.3 Possible PML technologies from other languages / domains

As mentioned, process modelling is a big and complex domain. There has been much discussion on the “right” PML. We must consider both technical and non-technical issues. Cf. also similar modelling discussions in other domains, where much relevant work has been done Which technology/theory to reuse? What has been reused?

Different process Modelling languages have their roots in computer science and related domains foundations. The main sources or approaches can be characterized as:

3.3.1 Project management

(359) Bar charts and activity networks are graphical notations that are used in project management. Activity bar charts describe which activities compose a project together with activity start and finish time. Staff allocation charts describe the binding between responsible per-

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. **Process analysts** have to provide an understandable, consistent, and verifiable model of the organisation process.
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. **Process designers** need to describe how the process model has to be implemented on top of a PSE. The process analyst and designer role is often merged into a process engineer role
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.**Process model programmers** have to code the above decisions in an executable PML
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 process agents (wet runs). This interaction occurs, respectively, through a tool interface (e.g. a BMS) and a user interface (e.g. through an agenda). **Process agents** need a somewhat similar view, though customized to their specific activities. **Project managers** need a project management view, because they need to understand, plan, and control project structure and work progress.
6. **Process Assessment** of quality and performance: 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.

(349) Well-known characteristics of any programming languages are:

(350) · **Formality**: syntax and semantics of a PML may be **formally**, precisely defined or **informally**, intuitively described. Formal PMLs support e.g. to reason about developed models, to analyze precisely defined properties of a model or to transform models in a consistent way.

(351) · **Expressiveness**: it depends from expressiveness of a PML whether all aspects of a process model may be directly modelled by language features of the PML or have, e.g., to be added by additional comments.

(352) · **Graphical** (often diagrammatically). This is closely related to the **understandability** of process models which is dependent on the possible users of a process model.

(353) · The PML may offer modelling-in-the-large concepts, as e.g. **abstraction** and **modularization** concepts, to structure a process model into submodels which are connected by certain relationships. Abstraction concepts may support the definition of more general,

(348) Three auxiliary process elements are the following. Note that tone could think of more auxiliary process elements.

11. **Cooperation model**: Cooperation protocols, sharing / locking etc need to be described.There are two basic modes of cooperation: *sequential*, e.g. by normal work or review chains, or *parallel*, e.g. upon workspace overlap. Cooperation model must include communication protocols of/on objects and coordination of actions (ordering and synchronisation). On a higher level of abstraction, it may include goal sharing and sophisticated interaction protocols such as negotiation in situations of conflict.

12. **Versioning/transaction model**: Versioning at least of the production workspace is needed, and likewise with some support for long transactions. The **transaction model** should be nested and allow pre-commit cooperation.

13. **Quality/performance model**: Overall quality model (product/process). A **product quality model** includes operational goals of product quality and associated metrics, e.g. review and test status [Naka90]. The **process performance model** for process quality expresses compliance to the stated process model, e.g. wrt. deadlines, budget, and user roles.

A process model consists of instances of these process elements together with additional constraints how they may be interrelated. Those interrelationship types are implicitly mentioned above, e.g., an activity *works-on* artifacts or agents ***play*** a role.

These basic elements and relationships allow a coarse-grained modelling of a software process. A software process modelling language has to provide language features to model these basic and advanced constituents (or sub-models) as well as their interrelationships. There exist a lot of variants how a PML supports the modelling of these ingredients of a process model. Each variant can be classified according to certain characteristics, which we recall in the following. As it is not possible to cover all characteristics within one specification language in the same depth, the process modelling language designer has to make a decision in case of trade-offs. This choice is closely related to the answers of the above mentioned question 2 (in which phase of a meta-process and who is dealing with the process model?), as it will be discussed later.

3.2.2 PML requirements and meta-process phases

A process model is used in different ways by different kinds of users during the different phases of the meta-process. Here, we mention the main meta-process phases together with their specific requirements.

1. **Process elicitation and requirement specifications**: during this phase, the **Process owner** needs to set overall goals, and understand the ramification of these by an abstract process model. 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.

4. **Human**: process agents and teams of such: An above role can be filled by humans with that role capability. A human **user** or process agent can fill a set of roles. He can also be a member of several **teams** or groups, possibly nested, representing either project teams or line organizations.

5. **Tool** for software production: This covers interactive as well as batch-wise tools, e.g. CASE design tools, document editors, and compilers. The tool details stand in the tool-view below. The tool model must specify how tools can be accessed and controlled. We must distinguish between batch and interactive tools, be able to handle call-backs from both. Batch tools cover compilers, links, and parsers etc. Interactive tools span from textual editors to graphic CASE tools

6. **Evolution Support**: General meta-process support for static or dynamic variability of the process model. 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).

(347) The four secondary process elements are:

7. **Project/organisation**: Organizations consist of humans, and have relationships to each other and to other process elements. Business rules and goals must be described here.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 ("Quality / Performance (auxiliary)" on page 11), cooperation patterns between projects, and connection to workspace transactions and versioning (page 11). Such project information is revised almost daily both to record ongoing work and to make adjustments based on this. General **goals**, e.g., business ones, have to be expressible.

8. **Work context**: A workspace (WS), containing and controlling artifacts for the actual (sub)process.These artifacts are often represented as **files** checked out from a repository. 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.

9. **Tool-view**: The tool view describes how much the process support has impact on the production process [Fugg94].

10. **User-view**: General user interface to help comprehend and guide the (enacting) process model. The external model representation may be different than the internal one. The problem of displaying complex and heterogeneous information to users with different levels of competence and goals, are shared by most computerised systems [Myer89]. 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). Some aspects to consider are: uniformity of presentation and interaction, easy comprehension of presented information, and flexible choice of presentation formats (filters, viewers).

and/or enacted within the meta process, as it has been discussed in chapter 2. Thus, any software process model has to model the real-world process appropriately and it has to meet the specific requirements of each phase of the meta process. These imply requirements for the process modelling language by which a process model is defined. These requirements on the process model and on the corresponding process modelling languages are discussed and explained within this section, which is organized according to the following questions:

1. What has to be defined within a process model? What are the constituents of a software process and how are they interrelated?
2. Which are **P**ML requirements and which is their relationships with each phase of the meta process and respective meta users?

3.2.1 Process elements

An appropriate model of a real-world situation consists of a description of the structure and behaviour of all problem-relevant constituents (i.e., objects) as well as of their static and dynamic interrelationships. The constituents and their relationships of models of a certain problem area can be classified into problem-oriented constituent and relationship. A concrete process model consists of instances of these constituent and relationship types. A software process consists of concurrent, cooperating activities, which cover software development and maintenance activities, as well as project management and quality assurance activities. First classifications of the constituents of a process model can be found in [19] , [Lon 93], [FH 93], [CFF 94]. They identify the following basic (or primary) constituent types:

The six primary process elements are:

1. **Activity**: A concurrent process step, working on software artifacts and coupled to a human role and to external production tool(s). It includes communication channels and control directives. Can be at different abstraction levels, i.e. decomposed activities. Concurrent, partly non-deterministic and cooperating activities are the heart of any process. These cover software development and maintenance activities, as well as project management and quality assurance activities, including meta-process ones. 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.
2. **Product**: A software artifact, persistent and versioned, simple or composite, with mutual relationships. The artifacts describe the product in question: software products and composites of them, associated documents, e.g. design documents, user documentation and test data. In a reflective system, all process model fragments can be considered artifacts, i.e. including “meta-data” like quality manuals and process models themselves.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.
3. **Role**: A role describes the rights and responsibilities of the human who will be in charge for an activity.

(336) These two dimentions set basic requirements for PML: each meta-process phase, e.g., elicitation must be supported by a language that enables to express the model at the appropriate abstraction level; each process element, e.g., tasks, roles, and products, needs to be described.

(337) There is a general agreement on identifying the primary process elements as activities (tasks), products, roles, tools, agents, and evolution support. Other elements, playing a secondary role in the context of a process model are work contexts, tool views, and user views. Finally, cooperation models, versioning and transaction, and quality models have to be described. In this chapter we distinguish among primary, secondary, and auxiliary elements.

(338) The big design challenges are questions are thus:

1. should we have a small kernel PML, and many specialized sub-PMLs for process subdomains? Or should we have one large PML, with sub-PMLs as views? And what about interoperability in federated systems, with sub-PMLs and related model repositories?
2. should one PML support each meta-process phase or should we have many PMLs, each devoted to a different meta-phase? Advantages and disadvantages stem from the respective ones in other domain modeling. If the same language is used across the different phases, then it is both easier to keep the different models consistent and and to reuse knowledge across the different phases. On the other hand, a single language is seldom sufficient to support the whole life cycle of a deliverable, from its conception to its implementation and operability.

(339) A PML can be **formal**, **semi-formal**, or **informal**. A formal PML is given with a formal syntax and semantics. Semi-formal languages usually have a graphical notation with formal syntax, but not a formal semantics i.e. not being executable. Natural languages, like English, may be used as informal PMLs. In this chapter, the emphasis is on formal PMLs, as they provide support for formal verification and analysis, simulation (dry runs), and execution (wet runs).

(340) A general software architecture for a PML comprises at least:

(341) · a repository in which the model is stored and maintained in a versioned fashion;

(342) · an intepreter to execute models;

(343) · an editor to facilitates model creation, and manipulation.

(344) We will discuss how the PML design choices influence the architectural choices, e.g., there can be several uncompatible PMLs, and each must be given with its own interprer, an editor, or there can be several compatible PMLs, in the sense that they can be executed by the same interpreter.

(345)

3.2 Requirements Process Modelling Languages

(346) A software process model is a representation of the real-world activities of a software production process. A software process model is developed, analyzed, refined, transformed

Chapter 3 Process Modelling Languages

(332) This chapter deals with the requirements and proposed solutions for process modelling languages, PMLs. Process Modelling is a very diverse and complex area. The needs for modelling and execution support are both technical (e.g. expressiveness, abstraction, and multi-perspectives) as well as non-technical (e.g. commercial support).

(333) There is obviously much existing research and technology, that potentially can be adapted for the process modelling domain. We should therefore look at related areas, such as information modelling, databases and groupware, for ideas and operative solutions.

(334) This chapter is structured as follows: Section 3.1 contains an introduction. Section 3.2 presents some requirements for PMLs. This includes the software process elements and process modelling phases that have to be described by a PML. It discusses views connected to process (meta-)roles, e.g. project managers, process engineers, process agents etc. It also deals with general modelling demands, such as understandability and modularization. Section 3.3 introduces the theoretical background for PMLs, by identifying the main conceptual sources and linguistic solutions for existing PMLs. Section 3.4 and Section 3.5 give a summary of existing PMLs, and classifies these wrt. linguistic paradigms and support offered. Some architectural of the associated PSEE are briefly discussed in Section 3.6. Some conclusions and directions for future research are given in Section 3.7.

3.1 Introduction

(335) As mentioned, a PML expresses software production processes in the form of a process model, being a computer-internal description of an external process.In the last 10 years, many PMLs have been proposed, implemented and tried out. However, there is little agreement or standardization, even on basic concepts and terminology. The first two chapters of this book constitute an attempt to formalize the basic concepts of the software process domain and the meta-process concept.