

Towards an Aspect-Oriented Approach to Improve the Reusability of Software Process Models

Rodrigo Quites Reis^{1,3} Carla Alessandra Lima Reis^{1,3} Heribert Schlebbe² Daltro José Nunes³

¹ Departamento de Informática, Universidade Federal do Pará (UFPA), Belém, PA, Brazil

² Fakultät Informatik, Universität Stuttgart (Uni-Stuttgart), Stuttgart, BW, Germany

³ Instituto de Informática, Universidade Federal do Rio Grande do Sul (UFRGS), Porto Alegre, RS, Brazil

URL: <http://www.inf.ufrgs.br/~prosoft>

E-mail: quites@computer.org, clima@ufpa.br, schlebbe@informatik.uni-stuttgart.de, daltro@inf.ufrgs.br

Abstract

The Aspect-Oriented (AO) paradigm is a recent trend to assist the development of high complexity software, proposing specific language level constructs to separate different concerns of software. This paper investigates the adoption of AO concepts to assist the design of high-level management policies in software process models. Process Policies are proposed in this text as first order modeling constructs to express syntactical and instantiation aspects of process models, complementing the role played by standard Process Modeling Languages (PMLs). The proposed approach is a consequence on the recent evolution in PML design to meet the recent advances in the Software Architecture field. Finally, the proposed model is briefly discussed with respect to its performance on describing standard software process examples, pointing to some required future improvements.

Keywords: Software Process Technology, Aspect-Oriented Software Development, Architecture Description Languages, Process Modeling Languages.

1 INTRODUCTION

Software is the base for a number of organizations involved in different activities around the planet, consisting on a strategic element to differentiate current products and services. Nowadays, software is embedded in a number of systems related to a wide selection of sciences and technologies, and according to Pressman [PRE01], software “will become the driver for new advances in everything from elementary education to genetic engineering”.

In spite of recent advances on Software Engineering, software community still discusses about the low quality and productivity of software industry, which have consequence on users’ satisfaction. In addition, requirements for new software applications increase in size and complexity, while managers expect to short the time to market. Thus, current software development processes are complex, involving a high quantity of specialized personnel that must work cooperatively among different locations, in an environment with technical and cultural challenges.

Software Process Technology (SPT) has been proposed to manage complex software development [FEI93]. Thus, tools, languages and methodologies are presented by the literature to improve industry-adopted software process models. Nowadays, SPT is profoundly influenced by the languages used during software process modeling, which are generically known as Process Modeling Languages (PML) [FEI99]. PMLs are used to formally describe software development processes through the precise specification of process tasks, components, and people influence on process results.

This text presents an investigation on the feasibility of providing separation of concerns to support the design of reusable process models. In addition, it analyzes the adoption of some ideas from the Aspect-Oriented (AO) paradigm in order to complement standard PML paradigms. The investigated hypotheses are:

- The concepts of Aspect and Crosscutting concerns are not tied to specific software paradigms (e.g., the Object-Oriented paradigm) or even to the software development domain.
- The complex, multiple viewpoint reality captured by software process models would greatly benefit from AO Programming concepts.

Therefore, this paper evaluates the feasibility on using policies as first order crosscutting properties for an activity-oriented PML called *APSEE*. The paper is organized as follows. Section 2 presents an overview on Software Process Technology. Section 3 briefly presents the underlying meta-model used for process modeling. Section 4 introduces the *StaticPolicy* and *InstantiationPolicy* languages. Section 6 discusses related work and presents the final remarks.

2 SOFTWARE PROCESS TECHNOLOGY

Competitive pressures on software industry have encouraged organizations to examine the effectiveness of their software development and evolution processes. According to [PAU94], a typical goal of an organization focused on achieving higher capability levels is to document software process and define one or more ideal processes to strive for.

Process-Centered Software Engineering Environments (PSEEs) constitute a special kind of Software Engineering Environments (SEEs) that support the rigorous definition of software processes, aiming to analyze, simulate, enact and reuse processes definition. Research groups and software industry developed many PSEEs during the last decade, including EPOS, ADELE, SPADE, PEACE+ and E³ (cited in [DER99]).

Complex software development involves activities performed by a number of developers with different technical

skills. Automated control is possible using a process model: many kinds of information are integrated in this model in order to identify “who, when, how and why the steps are performed during software development” [LON93]. Therefore, Process Modeling Languages (PMLs) are available to facilitate the description and manipulation of process models.

Software Process Reuse currently defines a wide area of research and practice related to different aspects of the reuse of knowledge obtained from previous successful projects. The specialized literature describes a number of descriptions of generic process models that can be reused in different contexts, varying from textual (narrative) descriptions to semi-formal process models. While informal process models are useful to describe methodologies for software development, the lack of rigor on the description of software process models inhibits their automation. Informal process models are also consequence of the lack of success on promoting adequate support for process modeling.

The current practice on describing reusable process models often relies on process designers' knowledge. Since the provocative proposal of Osterweil to evaluate the effectiveness of using standard software formalisms to describe software processes (“Software Processes are Software Too” [OST87]), the research on process modeling has been focused on adapting the existing solutions from (standard) product-oriented technology. In fact, recent studies highlighted the similarity of Architecture Description Languages (ADL) and Process formalisms [EST99]: both intend to offer high level language constructs (i.e. capable of providing a global picture of the process) that are executable (i.e. the communication between components can be automated).

Furthermore, while at a first sight the challenge of describing reusable process elements appears to be equivalent to the traditional software reuse problem, this is only partially true, since a process model mixes social, organizational, technological and environmental issues. The increasing complexity on software process modeling implies in the investigation on using successful general-purpose software development technologies in the process modeling field, as described next.

3 THE APSEE MODEL

Since the topic under discussion by this paper is a part of a larger work, where process technology is investigated in order to increase the level of automation for process management, this section presents the main characteristics of the proposed model to store reusable process descriptions and its underlying infrastructure.

3.1 The underlying APSEE meta-model

While a complete description and evaluation of the APSEE meta-model is out of the scope of this paper, this section presents an overview of the underlying infrastructure w.r.t. its historical roots and design characteristics.

APSEE is a software framework to automate software process management that evolved from a Software Process Engine [LIM98] originally proposed for the PROSOFT environment. PROSOFT is a formal object-based software development paradigm [NUN92] that is currently implemented as a Java-based SEE [SCH97] (in cooperation program between PPGC-UFRGS - Porto Alegre, Brazil - and Uni-Stuttgart - Germany). Nowadays, APSEE is the underlying integration kernel for a number of meta-models to support process simulation, instantiation, enactment and improvement and reuse [REI02a]. Figure 1 presents a couple of screenshots describing different views of an enacting process.

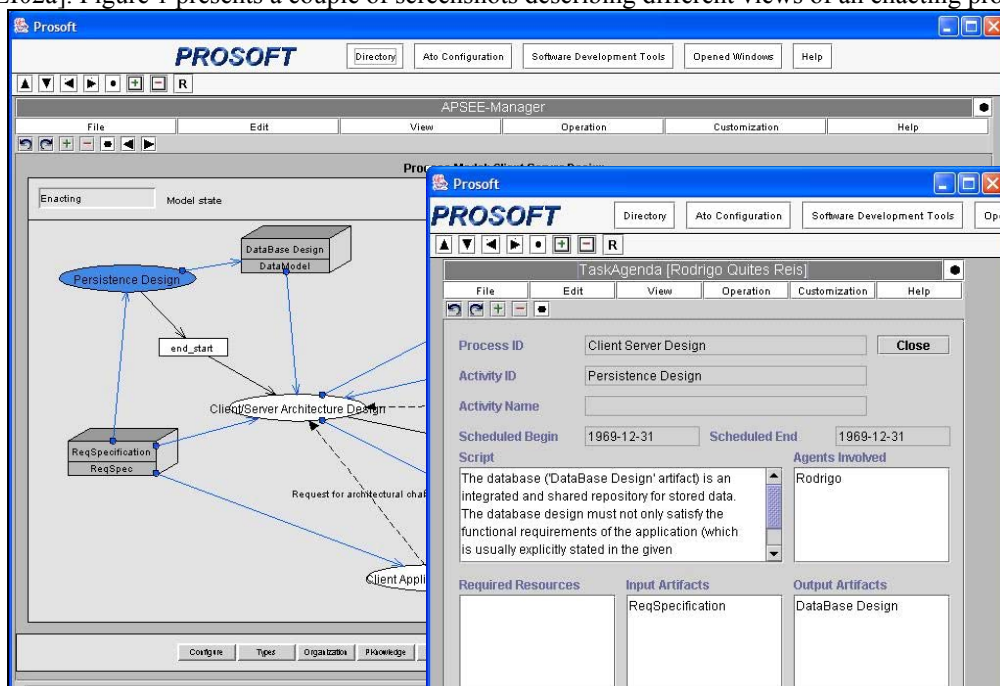


Figure 1 Snapshots of Manager (background – process monitor) and Developer (foreground - agenda) views of process enactment under the APSEE system

The *APSEE* meta-model is based on an activity-based paradigm, describing a process as a partially ordered collection of activities. A graph-based activity-oriented graphical PML is available (*APSEE-PML*), providing graphical representation for the proposed set of language constructs. A dynamic set of control mechanisms is available, delivering flexible synchronization on activity connections.

Three main process types are available, representing its engagement degree with details of instantiation of the model to its context. A **Process Template** supplies generic descriptions for posterior reuse. **Instantiated Processes** have information about its allocation. Finally, **Enacted Processes** store information about actual processes performance.

3.2 The *APSEE* System Architecture: separation of concerns for process models

The design of the *APSEE* architecture was profoundly influenced by the need to provide support for explicit separation of concerns during process modeling. According to [PER96], during the development of a process model its designer deals with a multi-dimensional reality that mixes information about the **Project** (i.e., instantiation details like organization-specific roles and detailed schedule), the **Environment** (the set of software tools needed to perform a development step/activity) and the **Process** itself. The proposed model extends the ideas of Perry by defining additional dimensions: **People** (including details about role types, agents, groups and abilities), **Software** (describing produced, consumed and transformed software artifacts) and **Resource** (describing consumable, exclusive and shareable resources).

The proposed *APSEE* Architecture is informally depicted as a set of layers in figure 2¹. The Processes layer is the most important since it defines processes and activities that are associated to components belonging to different layers during modeling and enactment time. Each layer describes elements that are defined independently (and are bound together during the construction of process models). It is important to note that the current system offers graphical representation in the PML notation only for Processes and Software components.

Figure 2 shows the connections among the enabled Policy instances. The solid lines describe to which process component the Policy instance is enabled. In the example, “A” and “B” constitute typical static policy instances (since they do not require additional information), while “C” and “D” instances constitute Resource Instantiation examples (since they are also associated to specific resource types). Finally, “E” is an example of an Agent Instantiation Policy: for a specific process activity it will apply some user-defined criteria to assist the selection of ‘Programmer’ agents.

The provided composition protocol is formally defined using a combination of algebraic methods (Graph Grammars and Algebraic PROSOFT [NUN92]) and it is further described in [REI02a].

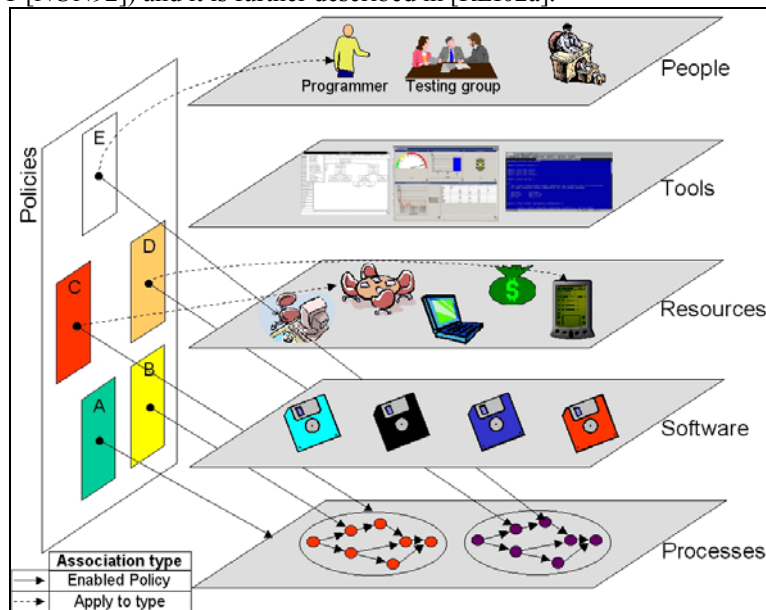


Figure 2 Orthogonality of Policies and Process Definitions

4 USING POLICIES AS ORTHOGONAL ELEMENTS FOR PROCESS MODELING

Process policies constitute “the guiding principles for process development and/or enactment” [FEI93]. In the *APSEE* model, the Policy constructor was designed to solve the need to store information about generic management strategies that would be difficult to express using standard (i.e., activity, rule or activity-oriented) PML constructs. Policies complement *APSEE-PML* by describing generic management strategies across different process components. Thus, this section highlights the influences from AO Paradigm to the design of the proposed Policies constructs.

4.1 The Aspect-Oriented Paradigm for Software Development

The AO paradigm has been proposed as a technique to improve the separation of concerns in software design and

¹ The Projects (i.e., Enacted Processes) layer is intentionally not presented in the layered diagram of figure 3, neither the relationship among the Tools, People, Resources and Software layers.

implementation. The main idea is that while the standard hierarchical modularity mechanisms provided by the OO paradigm are useful, they are inherently unable to modularize all concerns (properties) of interest in complex systems. Aspect-oriented software engineering provides mechanisms that explicitly capture the crosscutting system structure. Thus, the goal of AO software design is to support the developer in cleanly separating components (objects) and aspects (concerns) from each other, by providing mechanisms that make it possible to **abstract** and **compose** them to produce the overall system. Aspects are defined as properties that crosscut components in system's design.

Separating concerns from components requires a mechanism for composing – or *weaving* – them later. Central to the process of composing aspects and components is the concept of *join points*, the elements of the component language semantics handled by aspect specifications. Join points are well-defined points in the dynamic execution of the program. Examples of join points are method calls and receptions, method executions, and field sets and reads.

4.2 Process Policies

Figure 3 presents the correspondence of *APSEE* and AO constructs, w.r.t. the proposed policy modeling paradigm. The fundamental ideas that guided the recent evolution of AO and separation of concerns influenced the design of the *APSEE* Policy construct. The proposed approach for Policy definition, composition and enactment are similar to the ideas of Kenens [KEN98]: aspects (policies, in the adopted terminology) are available to handle static (syntax) and dynamic (instantiation) elements of the specific domain of process models.

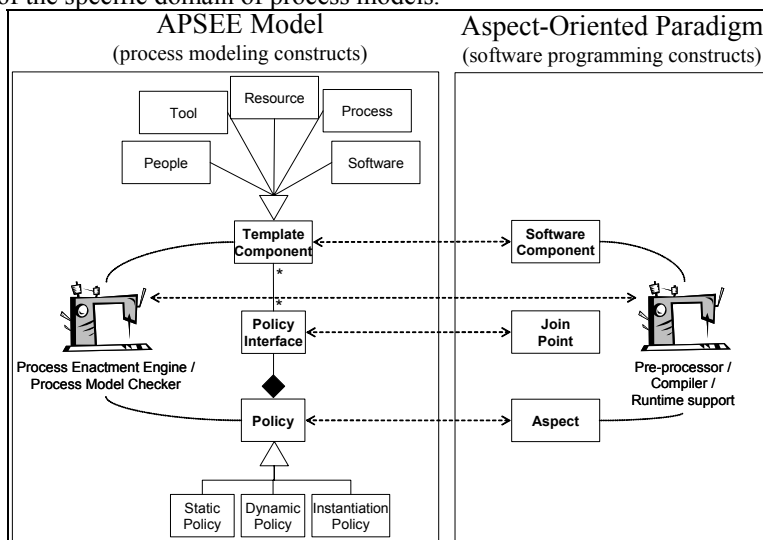


Figure 3 Mapping of concepts between the *APSEE* Model and the Aspect-Oriented Paradigm

Policies are active pluggable entities that restrict the syntactical formation of process models in response to static or dynamic process properties. Each Policy instance defines an Interface connector to the existing *APSEE* component types, establishing the required join points for the proposed approach. During enactment, the execution deviates from the normal process enactment flow to execute the associated policies, which are written with specialized languages. Three Policy types are available in the *APSEE* model: (Resource and People) Instantiation, Dynamic and Static Policies. The UML class diagram of figure 4 describes the main classes involved with Policies' definition. For the sake of brevity and clarity, this text presents next only the Static Policy and Resource Instantiation types.

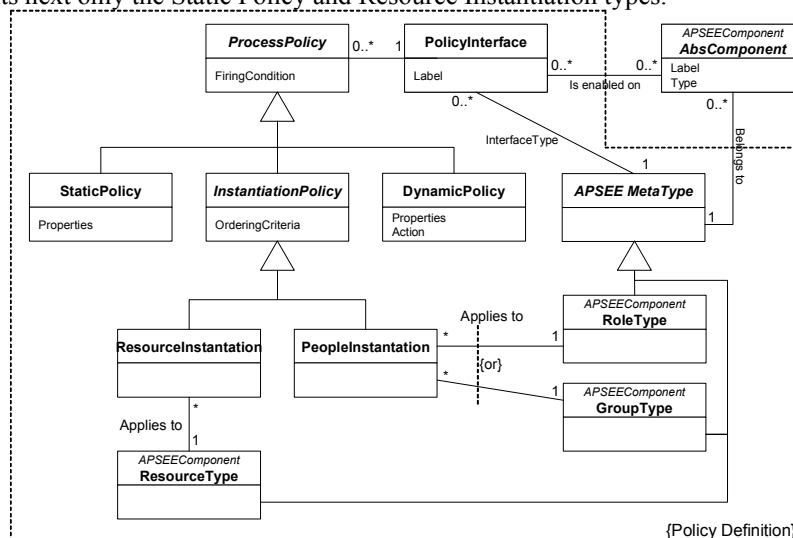


Figure 4 The Policies' data types

4.2.1 Static Policies

From a syntactic point of view, StaticPolicy main components are: its **name**, the **policy interface** (specifying the type to be observed), the **enabling properties** (logical pre-conditions to enable the policy – the *FiringCondition* attribute in figure 4), and the **properties** (an ordered list of logical conditions to be checked by *APSEE* before releasing a process model to enactment). Figure 5 shows a small example: it ensures that a Development activity must be followed by a Verification activity (involving different sets of agents), which, in turn, must have access to all artifacts manipulated by the development activity. The complete description of StaticPolicies semantics and additional examples are in [REI02a].

External Review: Static Policy Details	
Policy Interface	
Label	Type
a	<input checked="" type="radio"/> Activity <input type="radio"/> Resource <input type="radio"/> Agent
Enabling Properties	
a.get_type() sub_type_of "Development"	
Properties	
a.number_of_successors() = 1 and a.get_successors().all.get_type() sub_type_of "Verification" and a.get_successors().all.get_agents() not_contains a.get_agents() and a.get_successors().all.get_input_artifacts() contains a.get_output_artifacts() union a.get_input_artifacts() and a.get_successors().all.get_output_artifacts().any.get_type() sub_type_of "Review Report" and a.get_successors().any.is_source_feedback_to(a)	

Figure 5 The “External Review” static policy example

4.2.2 Instantiation Policies

Although the StaticPolicy construct is able to express some resource allocation restrictions that are verified during modeling time, the Instantiation offer a specific language to define ordering criteria for the dynamic selection of required resource instances. In fact, Policy enactment specialize the default resource instantiation algorithm available in the *APSEE* system: from required resource set in the abstract part of an activity, the proposed instantiation mechanism can automatically suggest an ordered candidate list based on the required resource types and their current (and expected) availability. Thus, Instantiation Policies constitute user-defined extensions to the default algorithm, with specialized restrictions and order criteria applied to specific process components.

Two examples are shown in figure 6: the leftmost example relies on dynamic (*is_late_to_begin* function call) and static (*get_type*) activity firing properties. Both examples as informally described in the respective Description field. The algebraic semantics for Resource Instantiation Policies is in [LIM01].

ID: "Late activity involving customer and meeting room" Name: "Late activity involving customer" Description: "If activity is late to begin and involves customer meeting, then take rooms that contains a beamer, with year utilization rate less than 10% and size smaller than 10m ² . Order them by highest MediumTimeBetweenFailure first" Interface: a:Activity; ApplyToType: "Meeting Room" Conditions: a.is_late_to_begin() and a.get_type() sub_type_of "Meet Customer" RestrictBy: Contains("beamer") getMetric("year_utilization_rate", <, 10) getMetric("size", <, 10) OrderBy: High_MTBF	ID: "Programming printers" Name: "Programming activities require inexpensive printers" Description: "If an activity belonging to the Programming type produce source code, then select inexpensive printers" Interface: a: Activity ApplyToType: "Printer" Conditions: a.get_output_artifacts().any.get_type() sub_type_of "SourceCode" and a.get_type() sub_type_of "Programming" RestrictBy: Cost(<, 10) NotRequires("Color Paper") OrderBy: Low_Cost
--	--

Figure 6 Resource Instantiation Policy examples

5 FINAL REMARKS

The recent AO Paradigm evolution showed that it can be successfully applied to different domains, including agent-based software engineering [GAR01] and formal methods [BLA98]. This text presented an overview on a novel aspect (policy)-oriented approach for software process modeling. Our experience showed that AO Paradigm seems to be a viable way to formally define critical management and enforcement strategies which are widely disseminated in the process patterns literature.

The proposed Process Policies constructs were successfully applied to a number of real world case studies (described in [REI02a] and [REI02b]). This approach improves the modularity on process model descriptions by providing independent formalisms. For instance, the modification on a policy instance changes the correspondent behavior for all the associated process components.

Few works in process modeling field are related to the description of aspects as first order components for process modeling. Perry proposes in [PER97] user-defined Policy descriptions for the Interact PML. Interact is a rule-based language which divides process description in three main parts: object, policy and activity definitions. Huang and Shan

proposal [HUA99] is similar to Instantiation Policy construct, providing a language to automate fine-grain personnel allocation in Workflow projects. *APSEE* Policies distance from the both cited proposals by encapsulating Policies as independently-defined pluggable elements that can be reused across different process models, as a consequence of adopting AO concepts for process modeling.

Both Static and Instantiation Policies were thoroughly used in the composition of a wide variety of process models, including literature-described and those representing new context-specific processes. This experience is summarized in [REI02a]. At a first sight, the use of the Policies construct would have the potential to increase the chance of conflicts. However, our initial experience showed that the Policies model scales well, since the Static Policies construct is able to detect potential conflicting policies in advance [REI02a], while the instantiation mechanism establishes an ordering algorithm that prevents firing of conflicting policies during instantiation time [LIM01].

It is important to note that the lack of a widely accepted standard notation for both AO software and process models constitutes a critical obstacle to the success of current solutions. In particular, we would like to evaluate the recent evolution of the work by Franch and Ribó [FRA99] (on extending UML to represent enactable process models), as it seems to offer a viable path for PML unification. Similar ongoing work in the context of extending the UML notation and semantics to deal with AO constructs would also be welcome.

Since *APSEE* is an active research project, some important model's components are still under implementation. An immediate extension to this work is to investigate the feasibility of using the proposed constructs on heterogeneous enactment and modeling systems – e.g., different PMLs - in order to take advantage of process designers' previous knowledge on different formalisms. Finally, the use of Policies to restrict the automated adaptation of process models to different contexts is currently under investigation [REI02a].

6 REFERENCES

- [BLA98] Blair, L.; Blair, G.S. The Impact of Aspect-Oriented Programming on Formal Methods. European Conference on Object-Oriented Programming (ECOOP'98) Proceedings..., 1998.
- [DER99] Derniame, J.; Kaba, B.; Wastell, D. (Eds.). Software Process. Lecture Notes in Computer Science 1500. Springer, 1999.
- [EST99] Estublier, J. Is a Process Formalism an ADL? Int'l Process Technology Workshop, 1. (IPTW'99). Proceedings... Sept. 1999.
- [FEI93] Feiler, P.; Humphrey, W. Software Process Development and Enactment. Int'l Conf. on the Software Process, 2. Feb.1993.
- [FRA99] Franch, X.; Ribó, J. Some Reflexions in Modeling of Software Process. Int'l. Process Techonology Workshop, 1. (IPTW'99). Proceedings... Sept.1999.
- [GAR01] Garcia, A. et. al. An Aspect-Based Approach for Developing Multi-Agent Object-Oriented Systems. Braz. Symposium on Software Engineering, 15. Proceedings... Rio, Oct. 2001.
- [HUA99] Huang, Y.; Shan, M. Policies in a Resource Manager of Workflow Systems: Modeling, Enforcement and Management. Int'l Conf. on Data Engineering, 15. Proceedings... Mar. 1999.
- [KEN98] Kenens P., et al. An AOP Case with Static and Dynamic Aspects. European Conference on Object-Oriented Programming (ECOOP'98) Proceedings... 1998.
- [LIM98] Lima Reis, C.A.; Reis, R.Q.; Nunes, D. J. Dynamic Software Process Manager for the PROSOFT Software Engineering Environment. Symposium on Software Technology (SoST'98). Proceedings... Sadio/European Software Institute. Sept.1998.
- [LIM01] Lima Reis, C.A. et al. A abordagem APSEE para Modelagem e Gerência de Recursos em Ambientes de Processos de Software. Brazilian Symposium on Software Engineering, 15. (SBES'2001) Proceedings... Oct.2001 (in Portuguese).
- [LON93] Lonchamp, J. A Structured Conceptual and Terminological Framework for the Software Process Engineering. Int'l Conf. on the Software Process, 2. Proceedings... IEEE CS, Mar.1993.
- [NUN92] Nunes, D. Estratégia Data Driven no Desenvolvimento de Software. Braz. Symp. on Software Engineering, 6. (SBES'92) Proceedings... Oct. 1992 (in Portuguese).
- [OST87] Osterweil, L. Software Processes are Software Too. Int'l. Conf. on Software Engineering, 9. Proceedings... IEEE CS. 1987.
- [PAU94] Paulk, M.; Weber, C.; Curtis, B. The Capability Maturity Model. Addison-Wesley Publishing Co., 1994.
- [PER96] Perry, D.E. Practical Issues in Process Reuse. Int'l. Software Process Workshop, 10. (ISPW'10) Proceedings... Jun.1996.
- [PER97] Perry, D.E. Using Process Modeling for Process Understanding. Software Process Improvement. Proceedings... Dec. 1997.
- [PRE01] Pressman, R.S. Software Engineering: A Practitioner's Approach. 5th European Edition. McGraw-Hill, 2001.
- [REI01] Reis, R. et al. Automated Support for Software Process Reuse. Int'l. Workshop on Groupware, 7. Proceedings... IEEE. 2001.
- [REI02a] Reis, R. et al. Automatic Verification of Static Policies on Software Process Models. Annals of Software Engineering. Special Volume on Process-Based Software Engineering. V.14. Kluwer Academic Publishers, Oct.2002 (to appear).
- [REI02b] Reis, R. et al. Towards a Software Process Model to Support the Design of Mobile Computing Applications. World Conference on Integrated Design & Process Technology, 6. Proceedings... Society for Design and Process Sciences, Grandview, TX, USA (to appear).
- [SCH97] Schlebbe, H.; Schimpf, S. Reengineering of Prosoft in Java. Technical Report. Uni-Stuttgart, Germany. Oct.1997.