Towards an Event Recording Mechanism for a Process-based Environment

Carla Paxiúba\textsuperscript{1,2}, Luciana Nascimento\textsuperscript{1,2}, Carla Lima Reis\textsuperscript{1}, Rodrigo Quites Reis\textsuperscript{1}

\textsuperscript{1} Laboratório de Engenharia de Software (LABES) - Departamento de Informática
\textsuperscript{2} Programa de Pós-Graduação em Engenharia Elétrica (PPGEE)
Universidade Federal do Pará (UFPA), Belém, Pará, Brasil
(paxiuba,luma,clima)@ufpa.br, quites@computer.org
http://www.labes.ufpa.br

Abstract : This paper discusses the initial experience on providing detailed event recording and management mechanism in a process-centered software engineering environment. The proposed model for event recording relies on the rigorous definition of the building blocks for process model composition that can influence process modeling, simulation, enaction and reuse phases. A Graph-Grammar based model was used to derive a Java-based prototype, which constitutes a first step towards a detailed knowledge base to support future advances in process mining technology. The results are discussed and highlight needs for future research in this topic.
1 Introduction

The Computer Science is one of the sciences that have been causing strong impacts in the current society. Software Engineering is an influential area, since software is universal and used in the majority of the organizations and in different activities, sciences and technologies. Thus, software has become a product and service differentiation factor. In this context, the quality has arisen as an essential element in software projects.

Competitive pressures on software industry have encouraged organizations to examine the effectiveness of their software development and evolution processes. According to [Paulk et al 1994], 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. In this context, Software Process Technology (SPT) has been proposed to manage complex software development [Feiler and Humphrey 1993]. Thus, tools, languages and methodologies are presented by the literature to improve industry-adopted software process models.

Process-Centered Software Engineering Environments (PSEE) constitute a special kind of Software Engineering Environments (SEE) that support the rigorous definition of software processes, aiming to analyze, simulate, enact and reuse processes definition. Research groups and software industry developed many PSEE during the last decade, including EPOS, ADELE, SPADE, PEACE+ and E3 (cited in [Derniame et al 1999]).

Event control in PSEE has acquired great importance; it allows better process management, which is a requirement for continuous quality and software process improvement. Thus, this paper presents a model of event management for a process-based software engineering environment - the APSEE environment.

The paper is organized as follows. Section 2 presents the APSEE model for automated process enactment support. Section 3 discusses the proposed events management model, while section 4 presents the final remarks.

2 The APSEE 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, its historical roots and design characteristics

2.1 The Underlying APSEE Meta-Model.

APSEE is a software framework to automate software process management that evolved from a standalone Software Process Engine originally proposed for the PROSOFT environment. Nowadays, APSEE is an open-source, underlying integration kernel for a number of meta-models to support process simulation, instantiation, enactment and improvement and reuse [Lima 2002][Lima et al 2002][Reis et al 2002]. Recently, some improvements have been proposed: the use Web Services for distributed communication; integration to open source and commercial tools (e.g., CVS, Eclipse, and Rational Rose); the adoption of a peer-to-peer protocol to support process enactment in virtual organizations.

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.

The *APSEE* design was influenced by the need to provide support for explicit separation of concerns during process modeling. A process designer traditionally 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 support an activity) and the Process itself. The adopted model extends this proposal by implementing additional dimensions: People (including details about role types, agents, groups and abilities), Software (produced, consumed and transformed software artifacts), Resource (describing consumable, exclusive and shareable support resources) and Policies (syntactical and instantiation aspects that constitute crosscutting properties to process components [Lima 2002][Reis et al 2002]). Each dimension describes independently defined components bound together during process modeling.

Figure 1 presents a couple of screenshots describing different views of an enacting process.

![Figure 1](image.png)

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

### 2.2 Graph Grammars

The complexity of software process enactment mechanism implies in the selection of a suitable formalism. The literature describes a number of experiences on using formal and semi-formal approaches with a variety degree of success. In general, most of the authors agree that formal semantics is an important requirement to specify the critical elements of process engines.

Graph Grammars (GGs) emerged in the 1970s as a promising alternative to
standard textual formal methods. GGs evolved as many methods and tools are available to support its use in a wide variety of domains [Bardohl 2000][Rozenberg 1997]. Regarding Process Technology, a remarkable experience is described in Dynamite [Westfechtel 1999], which applied a graph rewriting technique to specify its underlying meta-model.

Dynamite provides automated support for project planning, allows the enactment of partially incomplete process models, while provides a graphical PML. Dynamite design also includes a set of consistency rules. On the other hand, as stated in [Westfechtel 1999], it offers a limited set of control flow constructs (i.e., only end-start connections are supported). However, the experience on its design was valuable hint about the feasibility on using GGs to provide formal semantics for sophisticated process management mechanisms, in a graphical and intuitive way.

2.3 The APSEE-PML Visual Process Modeling Language

APSEE-PML is the graphical language used for process modeling in the APSEE environment. The APSEE meta-model includes information about the language components. A number of specification methods were used to specify APSEE types, including the Algebraic Method (for the definition of the required abstract data types) and graph grammars (for synchronization design).

The approach used for the specification of the APSEE visual process modeling language is called GenGED and was proposed by Bardohl in [Bardohl 2000]. In this approach, a visual language is specified by an alphabet and a grammar where the abstract and concrete syntaxes are distinguished. For the sake of brevity, this text does not include the algebraic specification for the required data types, which is presented in [Lima et al 2002].

![Diagram](image-url)

**Figure 2.** Graphical representation for the main APSEE Process Modeling Language constructs
The APSEE-PML graphical notation is summarized in Figure 2. Activities’ connections denote data and control flow as described next. Simple connections as associated to a dependency type (end-start, start-start and end-end, as proposed by [Ehrig et al 1999]). Simple Feedback connections are associated to logical conditions that enable the re-activation of a previous enacted activity. Thus, activity re-activation can be defined in advance (during modeling time), or manually included (enactment time) in response to a specific abnormal event (Re-activated activities are versioned, including the respective produced artifacts). Multiple Branch and Join connections are available in three types: AND, (inclusive-) OR and XOR. Finally, Artifact connections denote produced, consumed and transformed software artifacts.

3 The Proposed Event Management Model

Quality Models as the Capability Maturity Model for Software (CMM) [Paulk et al 1994] proposes the analysis of process results in order to facilitate both process and organization improvement. Knowledge from event logs can highlight weak and strong points in process and organization histories with the potential to feed decision support systems and process simulation tools. However, although there is little doubt about the benefits on providing automatic event management in PSEEs, the specialized literature presents no sufficiently detailed proposal on this topic. Thus, this section presents the event management model for the APSEE environment. Section 3.1 presents a motivation example that is later – sections 3.2 and 3.3 - used for the model description.

3.1 Motivation Example

Figure 3 presents a simple yet complete process model in APSEE-PML. In this example, the process begins with the Requirements Analysis (RA) activity. Due to the start-start connection to the Design activity (DSN), the process enactment start enables the start of DSN. The Design activity is responsible to produce the SW Design artifact, taking into account the Requirements specification (ReqSpec) provided by the first activity. The Implementation (IMP) activity does not depend on any other to begin, but its end is conditioned to the end of the Design activity (due to the end-end connection). Testing (TST) start is conditioned to the end of both Design and Implementation activities (due to the And Join end-start connection). The Testing activity makes use of an anonymous Resource and results on the Testing Report artifact. All activities in this example are associated to agents who are responsible for the actual process enactment (Jack, Anne, Joe, Bill and Kate).
Table 1 presents a simplified event log w.r.t. a fictitious enactment scenario for the given process model. It is important to note that for the sake of brevity, the events related to artifacts are omitted since they currently involve interaction with the CVS toolkit. Although Table 1 presents information derived from a simplified example, it does make use of the actual proposed structure for event recording.

The following sections provide detail on the explanation of Log attributes and the underlying mechanism.

3.2 The Log Package Data Model

The main element of the proposed event management model is the Log package, which consists of a set of classes and rules for event registration. Thus, this section is responsible to present the main data structure to store information about event occurrences.

As shown in the Figure 4, the Log package comprises all APSEE events. Its goal is to store information of what happened (What attribute), when it happened (When attribute), who was the responsible (Who attribute), and the reason of the event occurrence (Why attribute). The What attribute value is related to a specific instance of the event catalog (EventsCatalog class) or can store the identification of the firing rule that was the responsible for the occurrence.
Each software process in APSEE has a log (i.e., an instance for the Log class), that is composed by events (Event class). Events are, in turn, related to the following APSEE components: resources, process models, activities (global vision, developer vision and modeling events), connections between activities, and the software process itself.

Table 1. Event Log Example

<table>
<thead>
<tr>
<th>what</th>
<th>when</th>
<th>who</th>
<th>why</th>
<th>type</th>
<th>Id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ToEnacting</td>
<td>02/01/2005 08:00 am</td>
<td>Jim</td>
<td>Rule 1.1</td>
<td>ProcessEvent</td>
</tr>
<tr>
<td>2</td>
<td>ToReady</td>
<td>02/01/2005 08:00 am</td>
<td>APSEE Manager</td>
<td>Rule 2.1</td>
<td>AgendaEvent</td>
</tr>
<tr>
<td>3</td>
<td>ToReady</td>
<td>02/01/2005 08:00 am</td>
<td>APSEE Manager</td>
<td>Rule 2.1</td>
<td>GlobalActivityEvent</td>
</tr>
<tr>
<td>4</td>
<td>ToReady</td>
<td>02/01/2005 08:00 am</td>
<td>APSEE Manager</td>
<td>Rule 2.1</td>
<td>AgendaEvent</td>
</tr>
<tr>
<td>5</td>
<td>ToReady</td>
<td>02/01/2005 08:00 am</td>
<td>APSEE Manager</td>
<td>Rule 2.1</td>
<td>GlobalActivityEvent</td>
</tr>
<tr>
<td>6</td>
<td>ToReady</td>
<td>02/01/2005 08:00 am</td>
<td>APSEE Manager</td>
<td>Rule 2.1</td>
<td>AgendaEvent</td>
</tr>
<tr>
<td>7</td>
<td>ToActive</td>
<td>02/01/2005 08:00 am</td>
<td>APSEE Manager</td>
<td>Rule 2.1</td>
<td>GlobalActivityEvent</td>
</tr>
<tr>
<td>8</td>
<td>ToActive</td>
<td>02/01/2005 08:30 am</td>
<td>Jack</td>
<td>Rule 3.1</td>
<td>AgendaEvent</td>
</tr>
<tr>
<td>9</td>
<td>ToActive</td>
<td>02/01/2005 08:30 am</td>
<td>APSEE Manager</td>
<td>Rule 3.1</td>
<td>GlobalActivityEvent</td>
</tr>
<tr>
<td>10</td>
<td>ToReady</td>
<td>02/01/2005 08:00 am</td>
<td>APSEE Manager</td>
<td>Rule 2.1</td>
<td>AgendaEvent</td>
</tr>
<tr>
<td>11</td>
<td>ToReady</td>
<td>02/01/2005 08:00 am</td>
<td>APSEE Manager</td>
<td>Rule 2.1</td>
<td>AgendaEvent</td>
</tr>
<tr>
<td>12</td>
<td>ToReady</td>
<td>02/01/2005 08:00 am</td>
<td>APSEE Manager</td>
<td>Rule 2.1</td>
<td>GlobalActivityEvent</td>
</tr>
<tr>
<td>13</td>
<td>ToActive</td>
<td>02/01/2005 10:00 am</td>
<td>Anne</td>
<td>Rule 3.1</td>
<td>AgendaEvent</td>
</tr>
<tr>
<td>14</td>
<td>ToActive</td>
<td>02/01/2005 10:00 am</td>
<td>APSEE Manager</td>
<td>Rule 3.1</td>
<td>GlobalActivityEvent</td>
</tr>
<tr>
<td>15</td>
<td>ToFinished</td>
<td>02/05/2005 10:00 am</td>
<td>Jack</td>
<td>Rule 3.6</td>
<td>GlobalActivityEvent</td>
</tr>
<tr>
<td>16</td>
<td>ToFinished</td>
<td>02/05/2005 10:00 pm</td>
<td>APSEE Manager</td>
<td>Rule 3.6</td>
<td>AgendaEvent</td>
</tr>
<tr>
<td>17</td>
<td>ToFinished</td>
<td>02/05/2005 10:00 am</td>
<td>Anne</td>
<td>Rule 3.6</td>
<td>AgendaEvent</td>
</tr>
<tr>
<td>18</td>
<td>ToFinished</td>
<td>02/05/2005 10:00 pm</td>
<td>APSEE Manager</td>
<td>Rule 3.6</td>
<td>GlobalActivityEvent</td>
</tr>
<tr>
<td>19</td>
<td>ToActive</td>
<td>02/08/2005 4:30 pm</td>
<td>Joe</td>
<td>Rule 3.1</td>
<td>AgendaEvent</td>
</tr>
</tbody>
</table>
Table 1 refers only to the objects stored in the Log package itself. However, since the Id field contains references to existing objects from the environment metamodel, it is possible to navigate through the repository in order to collect additional required data. For example, it is possible to collect information about planned and actual dates and time required to accomplish a certain activity. In addition, it is possible to identify which agent and resource allocation policies where fired for a specific process model or activity. This is possible since each event type (represented by the specializations of the Event class in Figure 4), is associated to other APSEE elements described in [Paxiúba and Nascimento 2005].

The following summarize the functionalities for the main Log classes.
- **Event**

  *Event* is the main class to describe occurrences in a process Log. Each *Event* instance is related to a type of *APSEE* component (*EventType* class).

- **ProcessModelEvent**

  *ProcessModelEvent* is responsible to record the state changes for process models and decomposed activities. Each process is composed by a process model, which can be defined in a recurring manner (they are composed of activities that can be decomposed in new process models). Line number 32 is an example of *ProcessModelEvent* where a process model state change is registered in response to finish of its components.

- **GlobalActivityEvent**

  The *GlobalActivityEvent* class registers the global state changes of plain activities. In contrast to the *AgendaEvent* type (which stores the occurrences for an activity regarding a specific agent or group), the *Global* one is used only to store the changes related to consolidated actions. This control is required since an activity can be performed by several agents and groups, and the global state of the activity enactment can be different from the activity state in an agent’s agenda. A number of occurrences of *GlobalActivityEvent* are presented in table 1 (all of them with the GE sign in the *type* column). Most of *GlobalActivityEvent* occurrences in table 1 are consequence of internal process engine actions.

- **AgendaEvent**

  *AgendaEvent* handles the events in the developer’s (agent) point of view. The 11th line of Table 1 is an example of this occurrence that expresses the state change for the Design activity.

- **ConnectionEvent**

  Connections are used to establish links among activities in order to describe both process control and data flow. Connection events are registered in the *ConnectionEvent* class in accordance with the *APSEE* environment rules. Line 33 in Table 1 is an example of the recording of connection activation: in this case, it stores the firing of the AND Join Connection of Figure 4 in response to the end of the all preceding activities).

- **ProcessEvent**

  Process is a first order element in the *APSEE* environment, which is defined as a composition of all mentioned types. Thus, all process state changes in the *APSEE* environment are registered as events occurred in the *ProcessEvent* class.

  Table 1 shows two examples of *ProcessEvent* instances. The first line occurs in response to the demand of process start (invoked by its manager). Line number 42 is the second case, where a process state change is registered in response to finish of its components.

- **ResourceEvent**

  The *ResourceEvent* class stores the events regarding the management of resource state changes. The *APSEE* Resource model consists of resource types (Exclusive, Shareable and Consumable), resource instances, and relationships among them. Besides providing a mechanism to describe the available resources in the organization, the Resource model
is also used to control the competitive access to limited resources, to allocate them, to check their availability in the organization, and to verify the consistence of their use [9]. An example of ResourceEvent registration is in the 38th and 41th lines of Table 1.

- EventsCatalog

**EventsCatalog** class is a catalogue with all the events that can occur in APSEE environment for process components. This class also contains all the possible values for the **What** attribute.

- ModelingActivityEvent

Support for dynamic ad-hoc process changes is an important feature of APSEE [6]. Thus, a specific event type is associated to process model changes that influence its enactment. This justifies the **ModelingActivityEvent** class, which is not described here since dynamic process model change is out the scope of this paper.

### 3.3 Event Management Mechanism

The visual grammar of APSEE-PML is responsible for generating process model instances and enforcing their consistency when dynamic modifications are required, while behavior rules represent the execution semantics. More than 160 rules were described to handle process execution completely; a separate set of more than 200 rules were described for the generation grammar (which includes consistency checks) [Lima et al 2002].

It is important to note that the proposed Log package is a major improvement to the original APSEE as proposed by Lima Reis in [Lima et al 2002]. Ten (10) new rules were included in order to handle specific resource-related actions. In addition, more than 100 rules were updated to provide the proposed functionality.

Figure 5 shows Rule 1.1, as an introductory example. This rule, which corresponds to the **ExecuteProcess()** function call, describes the system reaction for a process initiation. In this case, the person in charge (called Manager in Figure 5) requests process execution (in the rule’s left side condition). If all conditions are satisfied (e.g., the process state is “not_started”), then a new **ProcessEvent** instance is generated (right side).

![Figure 5. Rule 1.1 – Process Initiation](image-url)

Figure 6 presents a rule (1.2) that is responsible to synchronize the developer-side agendas with the main process model. In this case, a process must be enacting and has an instantiated process model. The instantiated process model has, in turn, at least one null state normal activity (normal activities are performed by human beings – in
contrast to the automatic ones). The result, which is represented in the rule’s right side, places the activity in the waiting state, while the APSEE Manager (the system kernel) invokes CreateTasks to include references to it in the corresponding agendas. The updated version for this Rule – presented here – includes the creation of GlobalActivityEvent and AgendaEvent instances.

Figure 6. Rule 1.2 Agenda Synchronization

Figure 7 presents a rule (8.16) that register the instantiation of a Join Connection. Because the Join Connection is activated (Fired = TRUE at left side), the Join Or Connection is activated to and a new ConnectionEven instance is generated (right side).

Figure 7. Rule 8.16 Firing Connections

A complete rule description for event recording is available in [Paxiúba and Nascimento 2005].

4 Final Remarks

This paper presents a novel approach for event management in process-based environments. It is based on the notion that data preparation is an important precondition for successful data mining strategies [Pyle 1999]. In the context of software processes, the APSEE approach is also based on the idea of providing higher level of automation and flexibility to facilitate management.

It is important to point out some expected benefits from the proposed fully
automatic event management mechanism for a PSEE. First, it does provide log
information in a structured manner, which facilitates both the mining algorithms
themselves and human readability. Second, the event log is integrated to process and
artifact (CVS) repositories, allowing detailed observation of organizational data
relationships. Third, the collected information can provide a knowledge-based decision
support mechanism to automate different phases in the process lifecycle, which can
influence the advances on the current state-of-the-art of process simulation, diagnosis
and instantiation tools.

This proposal for event management had proved its effectiveness and
completeness through formal analysis and simulation. Currently, a number of
alternatives are considered to guide the final implementation of the Log package and the
set of updated rules for event management in the new APSEE system. For example,
provided rules could have been mapped to a set of triggers on a database management
system (which potentially inhibits portability across multiple DBMSs). The current
prototype implementation, however, is based on a mix of both object and aspect-
oriented constructs: new rules are mainly mapped to Java methods, while updated rules
are implemented through aspect-oriented modifiers to existing code.

Similar work on this topic has been proposed mainly in the Workflow field. For
example, Van der Aalst et al presents in [Aalst et al 2004] a process model discovery
approach from enactment data (i.e., there is no previous explicit process specification).
Process mining, in this case, is useful to discover hidden behavior patterns in order to
recover the actual process model.

In the near future, it is expected to conduct a detailed evaluation of the proposed
mechanism in the industry. This experience has the potential to provide feedback about
the key factors that influence process diagnosis and simulation.

5 Acknowledgments
This work is sponsored by the Brazilian Research Agencies - CNPq, CAPES and
FINEP.

6 References
Bandinelli, S., Fuggetta, A., Ghezzi, C. and Lavazza, L. “SPADE:An Environment for
Software Process Analysis, Design and Enactment”. In: FINKELSTEIN, A. et al.
(Ed.). Software Process Modelling and Technology. Tauton: Research Studies
Bardohl, R. GenGED - Visual Definition of Visual Languages based on Algebraic
Graph Transformation. PhD Thesis. Technische Universität Berlin. Kovac Verlag,
2000.
Derniame, J.; Kaba, B.; Wastell, D. (Eds.). Software Process. Lecture Notes in
Ehrig, H.; Engels, G.; Kreowski, H-J.; Rozenberg, G. Handbook of Graph Grammars
Lima Reis, C.A. “Resource Instantiation Policies in Software Process Environments”. 


