Journal of Software Engineering and Applications, **2012, 5, 375-384**
http://dx.doi.org/10.4236/jsea.2012.56044 Published Online June 2012 (http://www.SciRP.org/journal/jsea)

xSPIDER_ML: Proposal of a Software Processes Enactment Language Compliant with SPEM 2.0

Carlos Portela[1], Alexandre Vasconcelos[1], Antônio Silva[2], Elder Silva[2], Mariano Gomes[2], Maurício Ronny[2], Wallace Lira[2], Sandro Oliveira[2]

[1]Informatics Center, Federal University of Pernambuco, Recife, Brazil; [2]Institute of Exact and Natural Sciences, Federal University of Pará, Belém, Brazil.
Email: {csp3, amlv}@cin.ufpe.br, {aandrecunhas, elderferreirass, mauricio.ronny, wallace.lira}@gmail.com, {marianogomes, srbo}@ufpa.br

Received March 8th, 2012; revised April 3rd, 2012; accepted April 12th, 2012

ABSTRACT

SPEM (Software Process Engineering Metamodel Specification) is the software processes modeling standard defined by OMG (Object Management Group). However, the process enactment support provided by this standard has many deficiencies. Therefore, the main objective of this paper is to propose a language for software process enactment based upon SPEM 2.0 concepts. First, we will present a critical analysis of the SPEM standard approach for enactment. Then, we will present xSPIDER_ML, an enactment language, and describe its structure, components and associated rules. In order to evaluate the proposed language, a case study is performed through a RUP (Rational Unified Process) process instantiation. The language presented in this paper is part of a support set of tools for flexible software process enactment. Additionally, this set of tools is in compliance with software process quality models.

Keywords: Software Process; Enactment Language; SPEM; RUP

1. Introduction

Software process is the main object of study in Software Engineering and can be defined as the set of activities that aims to build software from a set of requirements [1]. The software development organizations must be able to define, use and improve their software development process. Thus, an organization must define a standard process which consists of a set of necessary tasks that can be instantiated in any software development project.

This standard process is a basis for defining the processes that are adopted for each project the company. These processes are known as instantiated processes that define how development projects will be executed [2]. It is necessary to use a modeling language for the construction of these process models. The OMG (Object Management Group) [3] defined the SPEM (Software & Systems Process Engineering Meta-Model) standard [2] for process modeling. The SPEM uses the notations of UML (Unified Modeling Language) [4] to define a specific set of stereotypes to support process modeling. SPEM 2.0 states to partially support the process enactment [2]. A major advantage of executable models is that they can be implemented, monitored, validated and improved [5].

However, the SPEM is the OMG standard language for process modeling, there are few organizations that use it [6]. Most process modeling tools are incorporated into PSEEs (Process-Centered Software Engineering Environments) being produced in academic environments and adopting its own language [5]. Furthermore, it was observed that only a subset of stereotypes and diagrams provided by SPEM is actually used by organizations that adopt this standard [6]. Moreover, the support the process enactment has many deficiencies [7]. Thus, the SPIDER Project [8], acronym standing for Software Process Improvement: DEvelopment and Research, opted for a definition of SPEM profile, called SPIDER_ML (Modeling Language) [9].

SPIDER_ML was developed to assist the modeling of software processes from a limited set of notations sufficient to model any software process [6]. However, SPIDER_ML does not offer native mechanisms for software process simulation and enactment similarly to SPEM.

This paper has two main objectives. The first objective is to introduce the SPEM 2.0 standard to support the enactment of software processes and to perform a critical analysis of this approach. The second objective is to present xSPIDER_ML that is an extension of the modeling language SPIDER_ML which allows the enactment of SPEM 2.0 processes.

In addition to this introductory section, this paper presents in Section 2 a critical analysis on enactment approa-

ches proposed by SPEM 2.0, highlighting its limitations to support the process enactment. Section 3 defines an extension of SPIDER_ML that allows the specification of executable process models, compliant to the SPEM 2.0 standard. A case study of a RUP Process instantiation [10] is presented in Section 4 to evaluate and ensure that processes can be enacted regarding the structure and rules defined in the xSPIDER_ML behavioral semantics. Finally, the conclusions and future works of this research are presented in Section 5.

2. SPEM 2.0 and the Process Enactment

The SPEM 2.0 is the OMG standard dedicated to software process modeling. It is defined both as a metamodel and as a UML 2 profile [4]. Its main objective is to provide organizations with the means to define a conceptual framework. It does so providing the concepts necessary for modeling, interactions, documentation, management and presentation of their methods and development processes [2]. This structure is shown through a meta-model, composed of seven packages, as shown in **Figure 1**.

Each of these packages is composed by a set of specific stereotypes for modeling processes. These stereotypes are used with other UML notation aiming the extension of the semantic of their elements and relationships. This extension must comply with the rules established by the SPEM standard [2]. In general, these stereotypes allow us to link a special semantics with instances of the UML elements. Therefore, the SPEM allows its users (process modelers) to utilize the UML, defining stereotypes that can be used in basic UML diagrams, including the class, package, activity, use case, sequence and state transition diagrams.

The *Core* package introduces classes and abstraction and sets the basis for all other metamodel packages. The building block of this package is the class *Work Definition* that generalizes any work in compliance with SPEM 2.0 [5]. The *Process Structure* package defines elements to represent models of basic processes through a stream of *Activities* with their *Work Product Uses* and *Role Uses*. The possibility of describing textually these elements (*i.e.* add properties that describe the element) is supplied by the *Managed Content* package, which provides the management concepts of textual description of process elements. An example of such is the *Guidance* class. The *Method Content* package, defines key concepts for the specification of elements such as *Roles*, *Tasks* and *Work-Products*. The *Process with Methods* package defines the set of elements necessary for integration of processes. These processes are defined through the intersection of concepts from the *Process Structure* and *Method Content* packages. The *Method Plugin* package provides mechanisms for the management and reuse of method and process content libraries. Finally, the *Process Behavior* package provides a way to relate the SPEM 2.0 elements with external behavior model, such as UML [4] or BPMN (Business Process Modeling Notation) [11].

Although the process enactment has been a main demand when it was issued the RFP (Request For Proposal) [12] for the definition of SPEM 2.0 specification, the specification approved did not satisfy the enactment requirements [7]. There are two suggested approaches for process enactment models [2]. In the following subsection, these approaches are described and we present some observations regarding the viability of each of these approaches.

2.1. SPEM 2.0 Recommendations for Enactment

In the first approach [2], the SPEM 2.0 standard proposes the mapping of its processes into project plans through project planning and enactment systems such as IBM

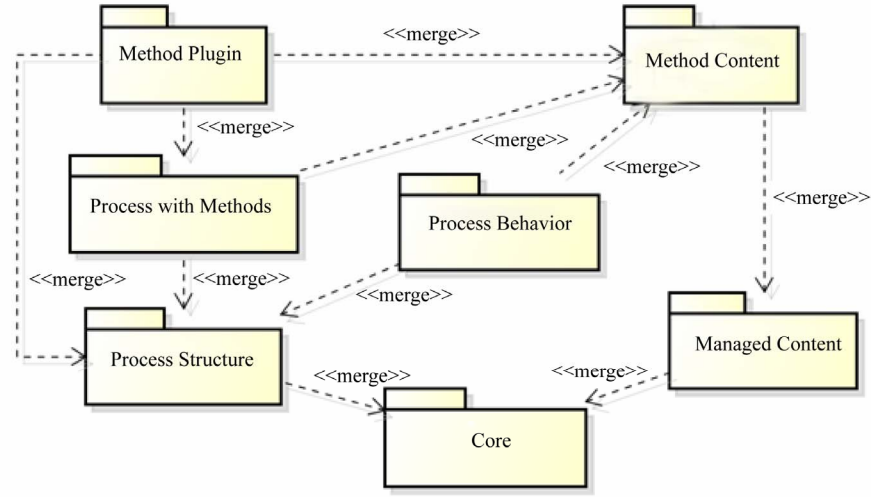

Figure 1. SPEM 2.0 structure [2].

Rational Portfolio Manager or Microsoft Project. After the mapping with project plans, these can be instantiated through planning tools where resources can be allocated. However, despite this approach being very useful for project planning, it does not meet the process enactment requirements that are: automatic changes in the tasks for responsible roles; automatic artifact forwarding; automatic control of the state of work products after each activity; among other features [7]. Besides, this approach biggest disadvantage is its strong dependence on a project planning tool.

The second approach [2] provides a way to relate SPEM 2.0 process elements with external behavior models, through the *Process Behavior* package. The objective is enable the process modeler the possibility of choosing an execution method that best fits their needs, in which case this approach authors claims greater flexibility. For example, the SPEM 2.0 elements can be mapped to the execution language BPEL (Business Process Execution Language) [11] aiming to reuse BPEL execution mechanism. This mechanism is a standard execution language in the business processes area.

However, this second approach offers flexibility regarding the representation of behavioral aspects of the SPEM 2.0 processes, it has some deficiencies. For instance, the standard is unclear about how the relationship of the process elements with behavior models must be addressed [7]. It just provides proxy classes that reference other elements in an external behavioral model. The second deficiency is that the mapping of SPEM 2.0 process elements on a specific model of behavior can be done differently in an organization depending on the interpretation of the process modeler. Thus, a standardization effort may be necessary in order to adjust the mapping rules between SPEM 2.0 concepts and a specific behavior model.

In order to address these deficiencies in the standard, there are some studies that propose to automate the enactment of software processes modeled using SPEM notations. These studies are presented in the following subsection.

2.2. Works Related to the SPEM Enactment

In [7], it is proposed a SPEM 2.0 extension called xSPEM (eXecutable SPEM) which provides the required concepts to enact a process model. A subset of the SPEM 2.0 notation is used, then other characteristics are added such as the definition of project assigned resources and activities to determine work product and task attributes. It also defines ways to store the process status during the enactment. Since both the process model and project model were defined, it is proposed to validate these models using formal methods, such as model-checking available in the Petri Nets area. Finally, it is presented the mapping rules between a subset of the SPEM 2.0 concepts and BPEL that allow them to be executed.

The approach presented in [13] proposes that the software development activities specified in SPEM should be transformed into a specification of the BPMN subprocess. This transformation is made through a Relations Language defined using the QVT approach (Query/Views/Transformations). Then the BPMN subprocess obtained is transformed into a language standard specification. It is necessary to define a process workflow language, such as BPEL4WS (Business Process Execution Language for Web Services) or XPDL (XML Process Definition Language). Afterwards, the procedure according to the selected language will be the input to a workflow engine such as Open Business engine (which supports XPDL) and BPEL Process Manager (which supports BPEL).

These approaches have deficiencies regarding the preparation and maintenance of process enactment, such as:

- The main SPEM elements that provide appropriate semantics for process modeling have no equivalents in BPMN. This causes the loss of appropriate semantics for software process modeling;
- The transformations between models are necessary for some refinement stages before they can be executed;
- These approaches require a great effort in maintaining the mapping between models in the case of any change in the process during enactment. This creates the problem of traceability and how these changes may impact on the SPEM 2.0 initial model.

Therefore, it is known that the SPEM standard does not provide by default any concepts or formalisms for process enactment [7] and that the approaches that allow the enactment of these processes from the models transformation define transformation specific standards that demand a great effort to keep the correspondence between the initial model (SPEM) and the final model (BPMN). In this scenario, we present a proposal in Section 3 of a process enactment language which aims to support the implementation of these SPEM models without the need to use transition models.

3. xSPIDER_ML: Processes Enactment Language

The SPIDER_ML language was created based upon a survey and an analysis of several modeling languages both performed in [14]. Furthermore, it was also based upon observations made during the implementation of organizational process improvement programs [6]. Thus it was established a set of the software industry most used practices regarding the processes definition and modeling. This language main objectives are [6]: to incorporate and to formalize the process modeling practices used by software industry; refining and reusing the

set of SPEM and UML elements; and to make the process modeling easier by using a reduced set of elements when compared with the number of SPEM elements. This language is adopted by the process modeling tool Spider-PM[1] [15], developed by the SPIDER Project, under GPL (General Public License).

SPIDER_ML can represent any organizational software process despite presenting a reduced set of elements [6]. This is possible because these elements are enough to represent the structures of Standard Processes and Instantiated Processes. These structures are described using UML diagrams, in which: the package diagram describes the Standard Process; the activity diagram represents the Instantiated Process; and the class diagram presents the associations between the elements.

The **Table 1** shows some elements of SPIDER_ML that refers to Instantiated Process. These elements will be used in the Section 4 case study. The details of these elements as well as the complete SPIDER_ML structure are available in its technical specification [9].

However, as mentioned in Section 1, both SPIDER_ML and SPEM have no native mechanisms for automated enactment of the software processes. Therefore, it is proposed the xSPIDER_ML approach (eXecutable SPIDER_ML). This is a SPIDER_ML extension created in order to allow SPIDER_ML modeled processes to be executed. For this purpose, we defined a formalism that added new components and attributes, as well as incorporating enactment rules to SPIDER_ML.

In addition, xSPIDER_ML aims to allow that the enactment is performed in a flexibly manner, semi-automated in compliance with the major process maturity models adopted in the Brazilian scenario: CMMI-DEV [16] and MR-MPS [17]. This compliance is related to the process capacity as specified in these two models. Process capability is defined as the refinement degree and institutionalization that the process is executed in the organization or in the organizational unit [17].

The xSPIDER_ML language is one of the Spider-PE tools. Spider-PE [18] is a support set of tools designed in order to support the software process enactment. The other tools that complement the Spider-PE include: 1) a best practices mapping between the CMMI-DEV and MR-MPS models. This mapping is performed considering the process capacity in order to identify which best practices related to the process enactment; 2) a framework that combines the practices identified in the mapping and in the xSPIDER_ML formalism enactment. This framework defines generic activities that can be adopted by any organization that wants to enact its software process; and 3) a free software tool that implements the framework activities and the xSPIDER_ML enactment formalism. This software tool was conceived in order to systematize the flexible and semi-automated process enactment. Furthermore, this tool is designed in order to validate the proposed language and framework.

[1]Available at http://spider.ufpa.br/index.php?id=resultados.

Table 1. Instantiated process elements [9].

Notation	Element	Description
	Process	A Standard Process instance for a specific project.
	Phase	A significant period of an Instantiated Process.
	Iteration	A set of Activities and Tasks that are repeated in a Phase.
	Milestone	A significant event in an Instantiated Process.
	Activity	A work to be performed during an Instantiated Process and that can be decomposed.
	Task Use	A work that cannot be decomposed.

The study presented in [5] was the Spider-PE main reference regarding the flexibility concept during enactment. This referenced study defines the conceptual basis for the WebAPSEE environment definition which uses a visual language called WebAPSEE-PML (Process Modeling Language). This environment adopts the formal specification with graph grammars approach to assist the enactment of the processes modeled in this language. In the SPIDER Project context, with respect to life cycle processes research, we developed a tool for software process simulation, called SPSM (Software Process Simulator Machine) [19]. This simulation tool adopts the SPIDER_ML syntax as basis for simulating the enactment of process models.

Based upon the premises presented in this section, the xSPIDER_ML was designed as an SPIDER_ML extension that adds to the later some components (described in Subsection 3.1) and implements rules (described in Subsection 3.2). Both characteristics combined add a behavioral semantic to the SPIDER_ML language (therefore adds to SPEM 2.0). This formal enactment only occurs on the elements that compose the Instantiated Processes structure (shown in **Table 1**). This occurs because these processes are enacted by an organization considering the resources available and the specific characteristics of software that will be produced [14].

3.1. Language Structure

Considering that the xSPIDER_ML objective is to enact SPIDER_ML models (characterized as SPEM profile),

we chose to define its structure based upon the structure proposed by xSPEM [7]. This choice was made because both approaches aim at making the SPEM 2.0 executable. The purpose of this packets structure is to provide organizations with the means to define a conceptual structure and the necessary concepts for the semi-automated enactment of their development processes. The xSPIDER_ML structure was divided in: xSPIDERML_Core, *Process Parameters*, *Project Variables*, *Event Types* and *Process Trace*.

For a better understanding, it will be presented only a subset of these language concepts, selected according to their relevance for understanding the case study (presented in Section 4). The details of the other concepts that compose the xSPIDER_ML are available in its technical specification [20].

The SPIDER_ML elements related to the Instantiated Processes (see **Table 1**) are grouped in the xSPIDERML_Core package. In addition, the xSPIDERML_Core package reuses concepts and xSPEM [7] and SPEM 2.0 [2] elements to provide all the necessary elements means for defining and structuring a software process suitable for process enactment. These elements determine the basis for all other xSPIDER_ML packages, as shown in **Figure 2**.

In this package we highlight the importance of the *Activity* component. This is a *Work Breakdown Element* and *Work Definition* specialization which defines the general unit of work within a process as well as a process itself. It relates to *Work Product Use* through *Process Parameter* class instances and *Role Use* through *Process Performer Map* instances. The *Process* class represents a set of work definitions partially ordered with the intention of achieving development goals, such as the specific software system delivery. These processes are characterized as *Phases* and *Milestones* sequences and expressing the product development life cycle.

Phase represents a significant period of time for a project. Usually there are events that occur in the phase end, such as a control point, a milestone or the delivery of a product to the customer. *Milestone* is a *Work Breakdown Element* that represents a significant event for a development project. Events usually occur in the *Milestone* such as an important decision-making or the delivery of a working version of the software. Iteration consists of a set of activities and tasks that should be exe cuted in a loop. At the iteration end a *Milestone* can occur. Finally, the *Task Use* class in this package was also highlighted. This class must provide information related to the resources involved during the enactment of a represented task.

The process will evolve from one state to another during the enactment. In this context, state means the situation in which the process is in relation to the enactment: not started, enacting, paused, finalized. Through these states, it is possible to determine the current stage of the project development. Thus, it is necessary to define concepts for the characterization of all these process states during the enactment. This is the *Process Parameters* package goal. Therefore, the *Process Parameters* pack age defines properties for the xSPIDERML_Core basic structural elements that enables the enactment. The concepts regarding the process states types (*Process State* and *State Type* classes), task types (*Task Type* class) and flexibility in process enactment (*Feedback Connection* class) originated from the WebAPSEE approach [5]. Furthermore, we used concepts of states related to the enactment time of process elements (*Time State* class) from the xSPEM approach [7]. Finally, this package used elements from the SPSM approach [19]. These elements regard the connection and specialization of the *Task Use* class (*Dependency Connection*, *Stochastic Task* and *Continous Task* classes).

These properties contained in the *Process Parameters* package can be classified as universal and existential. The universal properties are those which must be completed in each enactment. For instance, every activity must start and end; once an activity is finalized, it has to stay in this state. These states are defined in the *State Type* and *Process State* classes. The existential properties are those that must be true at least for an enactment. As an example, each activity must be performed in a time between the *expected Start Time* and the *expected End Time*.

Additional resources are necessary in order to adjust the process of a project. This involves to define specific properties for activity scheduling and resource allocation. These properties are introduced in the *Project Variables* package that covers: the concepts of classification and resource states by Web APSEE [5]; estimated and real required workload to perform SPEM tasks [7] through the *Process Performer Map* class and through *Resource* and *Task Use* attributes; and the *Human Resource* attributes were added to empower the expressiveness of the xSPIDER_ML language. In this package it is redefined the *Activity* class, adding to its definition the expected time interval during which an activity must be enacted (*expected Start Time* and *expected End Time*) and the real time in which this activity occurs (*real Start Time* and *real End Time*) so that comparisons could be made.

Based upon xSPEM [7] and WebAPSEE [5] approaches the states and transitions were defined. These allow the process enactment evolution through the *Event Types* package. These actions are triggered by events (*Start Task*, *Pause Task*, *Resume Task*, *Cancel Task*, *Fail Task* and *Finish Task*) modeled as specializations of the *Task Event* class (an abstract event that involves a target task).

Finally, we identified the need to record these events

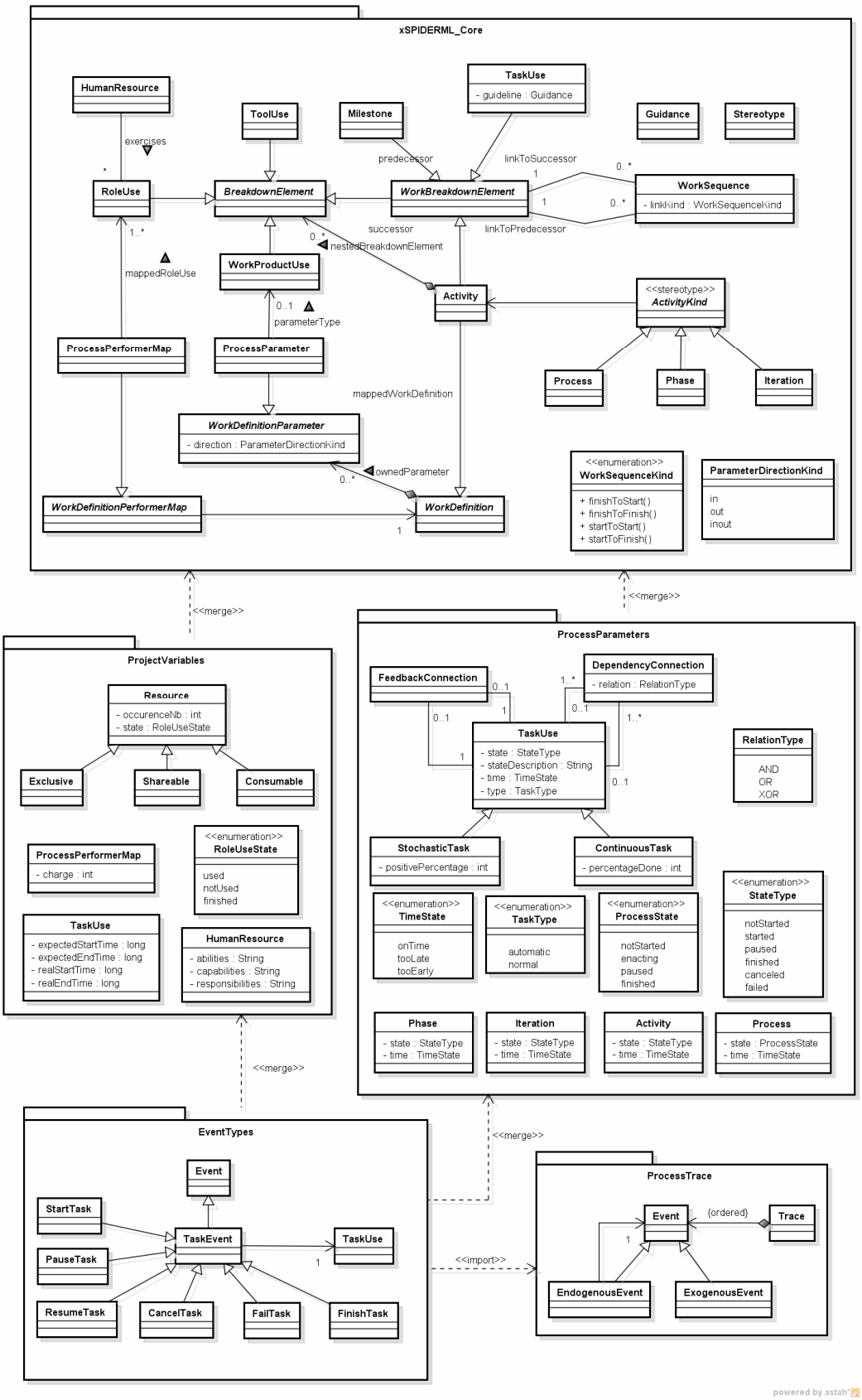

Figure 2. xSPIDER_ML packages structure [20].

performed during the process enactment that triggers transitions between states. We defined the *Process Trace* package based upon the xSPEM [7] because this approach has the same purpose as the approach proposed in this paper: to record the process enactment. These events may be produced by the process (*Exogenous Event*) or produced in the process (*Endogenous Event*).

3.2. Enactment Rules

After defining the xSPIDER_ML components structure, it was necessary to define rules to be applied on these elements and their relationships. In this context, rules define pre and post-conditions in a manner similar to an inference engine of a specialist system [5]. These rules allow you to extend the SPIDER_ML semantics (and consequently the SPEM 2.0 semantics), in order to represent dynamic information, inherent in properties defined in Subsection 3.1. This section presents only a subset of rules, selected according to their relevance to the case study, described in Section 4. Full details of these rules base comprising the xSPIDER_ML are available in its technical specification [20].

The rules that compose the xSPIDER_ML formalism relate to state and time transition of the process elements. To present these rules, formal specification will be used based upon the xSPEM [7] and SPSM [19] approaches. This specification indicates a step-by-step enactment rules application on defined scenarios. Thus, we have:

- ∀ws—represents the *Work Sequence* class instantiation (contained in the xSPIDERML_Core package), indicating the relation between process elements (base and predecessor);
- predecessor—represents a *Work Sequence* class attribute that indicates which element precedes the enactment of the base element;
- link type—represents a *Work Sequence* class attribute that indicates the relationship type between elements (these types are present in *Work Sequence Kind* class of the xSPIDERML_Core package);
- link to predecessor state—represents a *Work Sequence* class attribute that indicates the possible connection state between the related elements (these states are present in *State Type* and *Process State* classes of the *Process Parameters* package);
- clock—represents the internal clock associated to the concept of enactment time of a specific *Work Breakdown Element*.

According to the xSPIDER_ML structure, it is possible to identify two aspects common to the *Task Use* components. First, a task can assume the states: *not Started*, *started*, *paused* and *finished*. Secondly, there is a time sense and clock associated with each task that can be represented from the set {*too Early*, *on Time*, *too Late*}. In order to apply rules to these aspects, it is necessary to extend the *Task Use* element in order to introduce attributes that reflect the dynamic information (the current task state and the internal clock concept). This clock concept is abstract and not represented in the xSPIDER_ML structure. However, this concept should be taken into consideration by the enactment mechanism to adopt this language.

An abstract observation of the operational semantics of processes enactment in relation to these properties can be performed. Considering *t* as the task to be performed, whose initial state is *not Started*, the state transitions relations are presented in the **Figure 3**.

Attached to each task, there is the concept of internal clock to check its enactment time status. Thus, for a task *t* with initial state *started* and status *on Time* has the following possibilities are represented in the **Figure 4**.

4. Case Study: Small RUP Instantiation

In order to evaluate the enactment language proposed in this paper and to clarify the use of its components and associated rules, we present (from the modeling in the **Figure 5**) an instantiation of a RUP (Rational Unified Process) profile for small projects, available at [10].

The case study objective is to demonstrate that xSPIDER_ML language can be applied in a real process evaluating the enactment formalism proposed by this language. Supposing that the RUP Process instance (shown in **Figure 5**) consists of four phases: Inception, Elaboration, Construction and Transition. First, it is presented the model of this process using the SPIDER_ML notations (**Table 1**).

This process modeling was performed with the Spider-

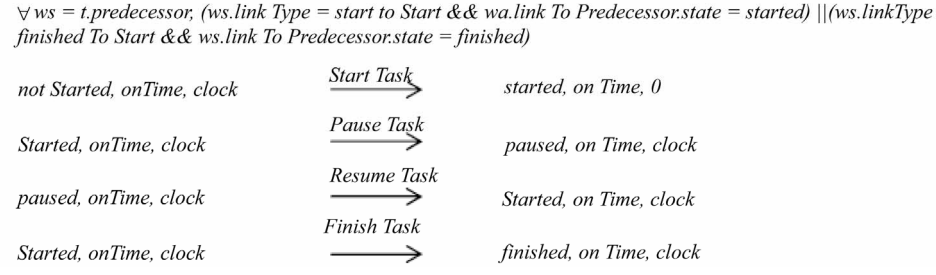

Figure 3. Transition states of a task.

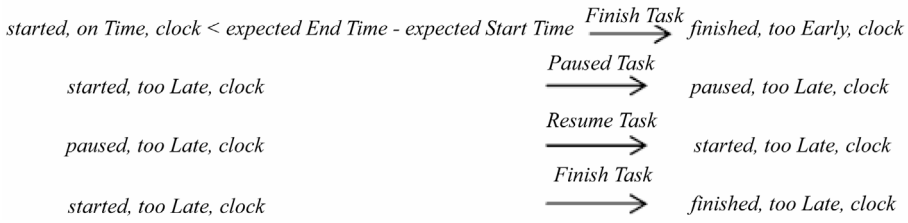

Figure 4. Transition status of a task.

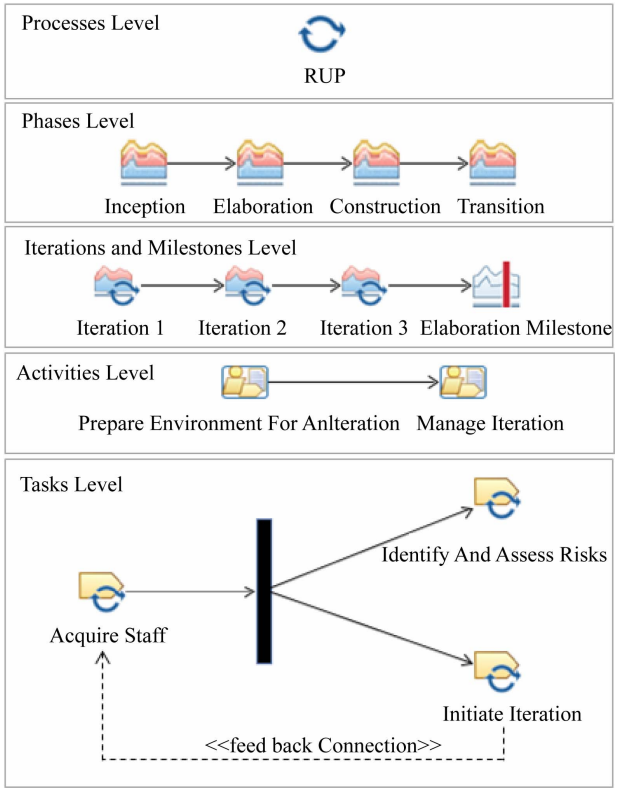

Figure 5. RUP process modeling.

PM tool [15]. Since this tool applies the process hierarchy settings, it does not allow the relationship between elements of different hierarchical levels (*i.e.* a direct connection between *Process Level* elements and *Phases Level* elements).

Assuming that this process is in the *Elaboration* Phase in a given moment of his enactment, we chose to adopt the UML object diagram [4] to represent this situation. This diagram allows to instantiate the classes defined in the *Process Parameters* package (Subsection 3.1), as shown in **Figure 6**.

For this process, enacting (state = *enacting*) according to the expected time (time = *On Time*), the *Inception* Phase was completed late in relation to the plan (state = *enacting* and time = *too Late*). The *Construction* and *Transition* Phases have not started (state = *not Started*).

The *Elaboration* Phase profile of RUP Process consists of three iterations. By the end of the third interaction the *Elaboration Milestone* occurs, from which only the first iteration (*Iteration* 1) was initiated (state = *started* and time = *on Time*).

For this iteration, the *Prepare Environment For AnIteration* activity is represented composed by the *Manage Iteration* activity. These activities are represented in the started state and in compliance with the expected time (state = *started* and time = *on Time*). This later activity is composed by *Acquire Staff*, *Identify And Assess Risks* and *Initiate Iteration* tasks.

Suppose that the *Acquire Staff* task was initially finished *on Time*. Subsequently, according to the classification of the dependency connection established between tasks (*AND*), the *Initiate Iteration* and *Identify And Assess Risks* tasks are performed simultaneously. In the example, the *Identify And Assess Risks* task was completed earlier than the expected time (time = *too Early*). However, there was a need to pause (state = *paused*) the *Initiate Iteration* task because it requires the allocation of more human resources for its implementation (state Description = "*Is Necessary to allocate more staff*"). The allocation of human resources was carried out before the *Acquire Staff* task. Therefore, in order to remove the obstruction in the *Initiate Iteration* task accomplishing, we used a Feedback Connection (FC1) to return to the process enactment. This return allows us to rerun the *Acquire Staff* task.

It is important to highlight that the case study presented in this paper has some limitations because it presents only a certain enactment stage and only instances of *Process Parameters* package elements described in Sub section 3.1. However, the full details of this case study are available in [20].

5. Final Thoughts

In this paper, we proposed an extension for the SPIDER_ML [9] language, which is characterized as a profile of the SPEM 2.0. This extension aims to provide the needed concepts to enact a process model. This enactment language proposed, xSPIDER_ML, adds elements and rules that allow the representation of dynamic information of these elements during the enactment, thus establishing a behavioral semantics so that processes modeled using SPIDER_ML notations can be enacted with

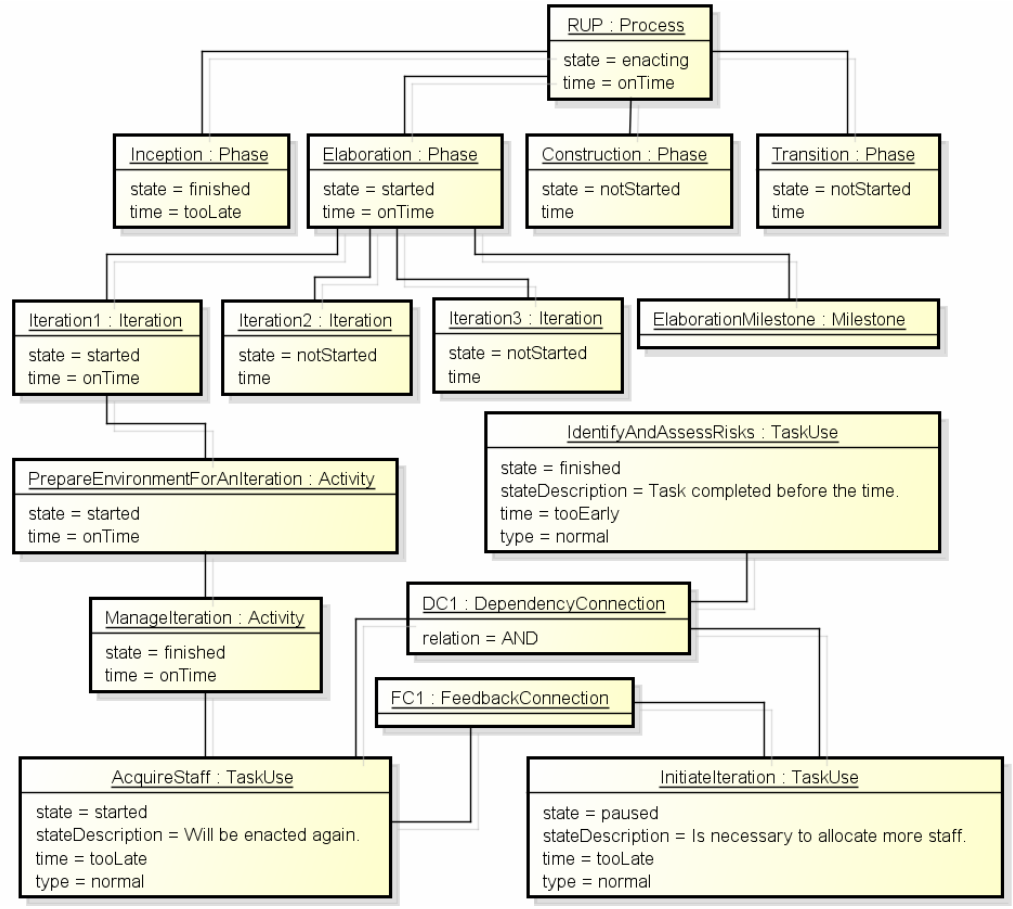

Figure 6. RUP process instantiation.

the aid of an execution machine. After describing the structure of components and rules, we presented a case study that instantiates a RUP [10] process. This case study, despite its limitations, allowed us to evaluate xSPIDER_ML from the use of its components and associated rules in a real process.

There are other approaches that allow the SPEM 2.0 processes enactment [7,13], as described in Subsection 2.2. However, these approaches have restrictions, such as: the appropriate semantics loss for software processes modeling in the transformation between models; the necessity of refinement steps before the process can be enacted; and a great effort in maintaining the mapping be tween models if any changes are made to the process during the enactment. These deficiencies are not present in xSPIDER_ML since it does not use any transformation process between models in its approach.

The xSPIDER_ML objective is to provide mechanisms that allow the flexible and semi-automated processes enactment in compliance with major quality models adopted in the Brazilian industry. As mentioned in Section 3, this language is part of a set of support tools for process enactment, proposed in [18]. In addition to this language, a Mapping between CMMI-DEV and MR-MPS Models and the Process Enactment Framework have been designed [21], which helped the language to address the compliance to best practices of these quality models. Currently, a free software tool that implements the language xSPIDER_ML from an execution engine is under development in the Project SPIDER laboratory.

6. Acknowledgements

The authors would like to thank CNPq (*Conselho Nacional de Desenvolvimento Científico e Tecnológico*— National Counsel of Technological and Scientific Development), for financial support through the DTI grant of the MCT/CNPq/FNDCT No. 19/2009 announcement for the development of this work.

REFERENCES

[1] W. Humphrey, “Managing the Software Process,” Addison-Wesley, Boston, 1989.

[2] http://www.omg.org/spec/SPEM/2.0/PDF

[3] http://www.omg.org/

[4] http://www.omg.org/spec/UML/

[5] C. Reis, “A Flexible Approach to Evolvable Software Pro-

cess Enactment," Ph.D. Thesis, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2003.

[6] R. Barros and S. Oliveira, "SPIDER_ML: A Software Process Modeling Language," 2*nd Escola Regional de Informatics*, Manaus, 6-8 October 2010.

[7] R. Bendraou, B. Combemale, X. Crégut and M. P. Gervais, "Definition of an Executable SPEM 2.0," 14*th Asia-Pacific Software Engineering Conference*, Aichi, 4-7 December 2007, pp. 390-397. doi:10.1109/ASPEC.2007.60

[8] S. Oliveira, *et al.*, "SPIDER: A Proposal for Systemic Solution of Free Software Suite Tools to Support Implementation of the MPS.BR Model," *Journal of the Brazilian Program of Quality and Productivity Software*, 2nd Editon, MCT/SEPIN, Brasilia, 2011, pp. 103-107.

[9] http://www.spider.ufpa.br/projetos/spider_pm/SPIDER_ML%5B1.1%5D.pdf

[10] http://www.wthreex.com/rup/smallprojects/index.htm

[11] http://www.omg.org/spec/BPMN/2.0/PDF

[12] http://www.omg.org/cgi-bin/doc?ad/2004-11-04

[13] F. Zorzán and D. Riesco, "Transformation in QVT of Software Development Process Based on SPEM to Workflows," *Journal Latin America Transactions*, Vol. 6, No. 7, 2008, pp. 655-660.

[14] S. Oliveira, "Software Process: Principles, Environments and Mechanisms Enactment," Ph.D. Thesis, Federal University of Pernambuco, Recife, 2006.

[15] R. Barros and S. Oliveira, "Spider-PM: Uma Ferramenta de Apoio à Modelagem de Processos de Software," 8*th Encontro Anual de Computação*, Catalão, 26-28 October 2010.

[16] http://www.sei.cmu.edu/reports/10tr033.pdf

[17] http://www.softex.br/mpsbr/_guias/guias/MPS.BR_Guia_Geral_2011.pdf

[18] C. Portela, A. Vasconcelos and S. Oliveira, "Spider-PE: A Tooling Support to Process Enactment adherent to the Quality Models," 9*th Workshop of Theses and Dissertations of Software Quality*, Curitiba, 6 June 2011.

[19] R. Chaves, E. Tavares, S. Oliveira and E. Favero, "A Software Process Simulator Machine for Software Engineering Simulation Games," *Brazilian Symposium on Games and Digital Entertainment*, Florianopolis, 8-10 November 2010, pp. 49-58. doi:10.1109/SBGAMES.2010.35

[20] http://www.spider.ufpa.br/projetos/spider_pe/xSPIDER_ML_Especificacao_Tecnica.pdf

[21] http://www.spider.ufpa.br/projetos/spider_pe/SPIDER-PE_Framework_Execucao_Processos.pdf

