A Model-Driven Proposal to Execute and Orchestrate Processes: PLM$_4$BS

Julián Alberto Garcia-Garcia(✉), Ayman Meidan, Antonio Vázquez Carreño, and Manuel Mejias Risoto

Web Engineering and Early Testing (IWT2) Group,
Escuela Técnica Superior de Ingeniería Informática, Universidad de Sevilla,
Avda Reina Mercedes s/n., 41012 Seville, Spain
{julian.garcia,antonio.vazquez}@iwt2.org,
ayman.meidan@gmail.com, risoto@us.es

Abstract. Business Processes Management (BPM) is a widely consolidated business strategy to improve and optimize the internal operation of any company. However, BPM is not usually simple to apply in software organizations because Software Processes (SPs) involve high degree of creativity, abstraction and rework, among other aspects. This situation provokes that these companies usually focus on modeling their processes but later, the orchestration and execution are manually and/or unilaterally performed by each involved role. This situation makes each SP difficult to maintain, monitor, evolve and measure. At present, there are model-based proposals to model SPs, but most of them fail to define the execution context of the process. This paper presents PLM$_4$BS, a model-driven framework to support modeling, execution and orchestration of SPs. It has been successfully validated in different real environments, what has returned us valuable feedback to improve PLM$_4$BS in the near future.

Keywords: Business Processes Management · Model-Driven Engineering · Execution and orchestration of processes

1 Introduction

It is a worldwide accepted knowledge that in the last years, Business Process Management (BPM) [1] has become a suitable strategy to increase excellence and productivity in any kind of organization. BPM tries to strategically assess processes and improve their effectiveness and efficiency within the organization with the aim to reduce costs and improve quality, productivity and competitiveness in relation to other organizations of the same business area.

Model Driven Engineering (MDE), aims to raise the level of abstraction in program specification and increase automation in program development. The idea promoted by MDE is to use models at different levels of abstraction for developing systems, thereby raising the level of abstraction in program specification.

However, although BPM has been successfully applied to many kinds of organizations, there are difficulties in software companies because of some special features of the software process. In [3], the authors identify and describe properties that characterize software processes in comparison with other processes (e.g., industrial processes) such as: (*i*) they are constantly evolving, as they usually incorporate new lifecycles and technologies and they frequently comprise several iterations that produce different software products versions; (*ii*) they are complex because they are strongly influenced by many unpredictable circumstances and many work teams; and (*iii*) they often rely on communication, coordination and cooperation of different frameworks and development technologies as well as on the different roles they play.

These features frequently provoke that BPM is not properly applied to software organizations that usually and justly focus on defining their processes, forgetting the process execution because most of the activities cannot be easily and effectively automated [4]. Once the process is defined, each involved role performs the process execution [5] and orchestration [6] manually and/or unilaterally. This statement describes a real situation that many software organizations are facing in their day-to-day lives. In fact, our research group has obtained this useful feedback from many partners (international and Spanish software companies) after carrying out many R&D projects.

The situation described previously poses other collateral problems. For instance, it may involve difficulties to manage, maintain, monitor, evolve and measure processes.

This paper aims to propose a MDE-based solution to support process execution and orchestration. For this purpose, this paper extends another previous paper [7] in which a process definition metamodel is presented within PLM$_4$BS (Process Lifecycle Management for Software-Business) framework. PLM$_4$BS is based on a continuous improvement lifecycle in order to manage the software process. This lifecycle defines four phases (modeling, execution and orchestration, monitoring and continuous improvement), although hitherto, PLM$_4$BS only supported the modeling phase [7].

This paper uses MDE (Model-Driven Engineering) [8] to integrate the execution and orchestration of processes within PLM$_4$BS because: it (*i*) is one of the most entrenched paradigms within software engineering area; and (*ii*) suitable results have been achieved when MDE has been applied to real environments (e.g., testing [13], healthcare environments [17] or Web engineering [16, 29], among others).

To achieve the aforementioned goals, this paper defines: (*i*) a specific process execution and orchestration[1] metamodel that lets specify the executable context of software processes after defining the process in the modeling phase; and (*ii*) a systematic and automatic protocol that makes it possible to generate executable code from an execution and orchestration model. This executable code is based on two standards: WS-BPEL (Business Process Execution Language for Web Services [9]) and XMI BPMN 2.0 format [10]. Both standards have been chosen because they are supported by most process engines (named BPMS or BPM Suite) according to conclusions obtained from different studies, such as [11]. This way, we are able to improve applicability of PLM$_4$BS to real environments since if an organization wants to use our proposal, it does not need to change its BPMS.

[1] The process orchestration is understood in this paper as the centralized coordination of events that allows conditioning the evolution and execution of process flow.

This paper is organized as follows: after this introduction, Sect. 2 analyzes the related work on model-based proposals to execute and orchestrate processes. Section 3 introduces our background and Sect. 4 describes our model-driven solution to execute and orchestrate processes. Finally, Sect. 5 presents some discussions, conclusions and future work.

2 Related Work

After modeling a process using a specific Process Modeling Language (PML) [18], it is necessary to define the execution context in order to perform and orchestrate it. The scope of this paper is framed into a model-based PML that includes mechanisms for defining this execution context. Nowadays, there are few proposals with some degree of executability. UML_4SPM [19] is a MOF-compliant metamodel to model software processes. Authors also propose to combine UML_4SPM and BPEL in order to execute the process.

Ferreira's proposal [20] consists in an UML-based modeling language to design software processes and a set of transformations rules to transform these UML process models into executable code. This code is implemented conform to Little-JIL [21], which is an ad-hoc, executable, programming graphical language to coordinate and run tasks among autonomous systems.

Di Nitto et al. [22] suggest an UML1.3-based framework to model SPs. However, they neither extend the UML metamodel or stereotypes nor introduce new concepts. The authors define transformation rules of a small subset of UML to generate executable workflow models. Later, these models can be deployed in an ad-hoc workflow management system [23] developed by these authors.

Chou's approach [24] uses activity diagrams of UML1.4 to model processes and establish theoretical transformation rules to generate executable code from activity diagrams. This code is implemented following an ad-hoc object-oriented programming language. The main disadvantage of this approach is the lack of an automatic generation of code from activity diagrams, what provokes that developers have to rewrite their software applications according to Chou's language.

Moreover, there is a standard proposal focused on software domain: SPEM2.0 [25]. SPEM2.0 is a standard that describes an UML-based metamodel that is used to define software development processes and software systems. However, SPEM2.0 does not provide mechanisms to execute the process. For this reason, Bendraou et al. propose xSPEM (eXecutable SPEM) [26], which provides a definition of an executable SPEM based on Petri-net. xSPEM adds some features to model and store states of the process when this one is executed.

3 Background

This section describes the context of PLM_4BS and its architecture (Sect. 3.1), which is based on MDE and a complete lifecycle. This paper focuses on supporting the second phase of the lifecycle of PLM_4BS (named "Execution and Orchestration Phase").

However, our proposal uses defined information in the first phase (named “Modeling Phase”). Consequently, a brief description of this first phase is also presented as background (Sect. 3.2).

3.1 MDE-Based Architecture of PLM$_4$BS

BPM can be considered a management strategy with a clear multidisciplinary nature that has conditioned the appearance of different views, definitions and perspectives of the process lifecycle and continuous improvement. However, orchestration of processes is an aspect that has not been clearly defined [1,27,28]. This is relevant because over the last decade, more companies used different interconnected tools to run their processes [30]. Therefore, it is necessary and important to support this feature, in a theoretical way, in the continuous improvement lifecycle of processes.

Considering the aforementioned arguments, the architecture of PLM$_4$BS includes a BP lifecycle comprising four phases: (1) modeling, (2) execution and orchestration, (3) monitoring and (4) continuous improvement. They are integrated within PLM$_4$BS using the MDE paradigm in order to take advantage of the benefits this paradigm entails [8].

Figure 1 shows conceptually this lifecycle of PLM$_4$BS as well as the phases that are completely and incompletely defined at present. The former (i.e., completely defined phases) are represented using a continuous line (these are: *(1) modeling* and *(2) execution and orchestration*) whereas the latter are represented by means of dashed lines (these are: *(3) monitoring* and *(4) continuous improvement*). It is important to point out that this paper is focused on describing the second phase of our lifecycle (i.e., execution and orchestration phase). The modeling phase (the first one) is briefly described in Sect. 3.2 as background because it constitutes an input to the execution and orchestration phase. Moreover, the third and fourth phases are conceived as future work, even though we are currently working on them. Finally, the phases of our process improvement lifecycle are further described below:

1. **Modeling Phase.** At this phase, the process engineer is able to model and describe his/her processes in a structured manner. PLM$_4$BS proposes a simple, flexible and highly semantic metamodel to support this phase. It is explained in Sect. 3.2 and takes the form of a MOF-compliant metamodel.
2. **Execution and Orchestration Phase.** Today, this phase is critical and essential since companies are being driven by the need to extensively automate their processes to execute and orchestrate them with EMS (Enterprise Management Systems). At this phase, the process defined by the process engineer at the previous phase must be executed and orchestrated in a BPMS. For this purpose, the process engineer must specify execution parameters as well as parameters for the communication and integration with external systems.

Nevertheless, most BPMSs have inflexible PMLs, that is, these tools do not allow executing processes that have been defined following other PMLs [11]. To solve this situation, PLM$_4$BS provides MDE mechanisms based on three steps.

On the one hand, PLM$_4$BS defines an ***execution and orchestration metamodel*** that defines execution parameters to run the process into a BPMS. Any instance of this

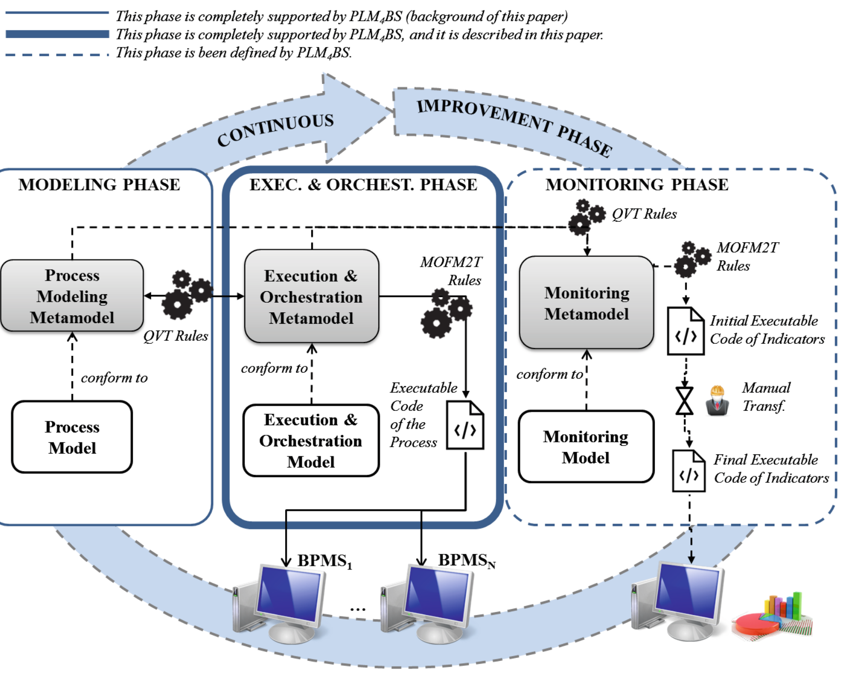
This phase is completely supported by PLM4BS (background of this paper)
This phase is completely supported by PLM4BS, and it is described in this paper.
This phase is been defined by PLM4BS.
CONTINUOUS
IMPROVEMENT PHASE
MODELING PHASE
EXEC. & ORCHEST. PHASE
MONITORING PHASE
Process Modeling Metamodel
QVT Rules
conform to
Process Model
Execution & Orchestration Metamodel
MOFM2T Rules
conform to
Executable Code of the Process
Execution & Orchestration Model
QVT Rules
MOFM2T Rules
Monitoring Metamodel
Initial Executable Code of Indicators
conform to
Manual Transf.
Monitoring Model
Final Executable Code of Indicators
BPMS1
BPMSN
...

Fig. 1. Theoretical architecture of PLM$_4$BS based on MDE and a continuous improvement lifecycle of processes. This paper focuses on describing in detail how PLM$_4$BS supports the execution and orchestration phase (second phase).

metamodel is systematically obtained using model-to-model (M2 M) transformation rules from the process modeling metamodel (modeling phase). Section 4 describes some of these rules, which are formalized using QVT Query/View/Transformation [31].

On the other hand, a ***systematic and automatic transformation protocol*** has been defined to generate executable code from the mentioned execution metamodel. This protocol is based on model-to-text (M2T) transformation rules using MOFM2T [32]. The process engineer should be able to instance and run processes into any process engine when the process execution context is defined.

All these mechanisms to support the execution and orchestration phase are explained in detail in Sect. 4.

3. **Monitoring Phase.** Once the process is deployed into a BPMS, it is time to evaluate its effectiveness. This evaluation provides a granular view of the overall productivity of each process and it is based on the definition of key performance indicators.

In this case, PLM$_4$BS provides two types of mechanisms to back up this phase. Firstly, the process modeling metamodel includes concepts (such as metric and indicator) that help the process engineer measure processes. Indicators are defined during the modeling phase. Secondly, and after identifying each indicator, PLM$_4$BS will define a monitoring metamodel that will include elements to allow the process measurement.

PLM$_4$BS plans to generate this monitoring metamodel from metamodels defined into previous phases. For this purpose, a set of M2 M transformation rules will be defined in PLM$_4$BS. Finally, a set of M2T transformation rules will be also defined in order to generate a measurement database and code scripts to manage each defined indicator. At present, this phase is not fully supported, thus, we are researching into different alternatives.

4. **Continuous Improvement Phase.** Finally, after evaluating processes performance (through assessment indicators and metrics), an organization should start an internal improvement process to achieve higher quality, efficiency, effectiveness and performance levels during processes execution. If necessary, the organization can iterate over our BPM lifecycle as many times as necessary in order to achieve business goals.

3.2 Metamodel to Support the Modeling Phase

PLM$_4$BS proposes a flexible and highly semantic metamodel (based on UML2.5) to support the modeling phase. This metamodel takes the form of a MOF-compliant metamodel and follows the guidelines defined in ISO/IEC TR 24744 standard [33]. Our Process Modeling Metamodel (PMM) will not be explained here, since it is out of the scope of this paper and it would become too extensive (a complete description can be found in [34]). However, a brief description of the main metaclasses of PMM is described below (Table 1). These metaclasses are the most important metaclasses to obtain the Process Execution and Orchestration Metamodel (PEOMM). It is also worth highlighting that our PMM contains other metaclasses (such as «Product», «Stakeholder» or «Indicator», among others), but they are not relevant for this paper.

Table 1. Main metaclasses of the modeling metamodel of PLM$_4$BS.

Metaclass	Meaning
Process	It represents any process that is composed of a set of ordered elements (i.e., *«ProcessElements» metaclass* linked themselves) to produce products («Product» metaclass)
ProcessElements ControlElement Activity HumanActivity OrchestrationActivity ComplexActivity	It represents any element of the process workflow and has been specialized in two metaclasses: *«ControlElement»* and *«Activity»* The former defines elements that allow establishing the process structure using different kinds of control elements: *«InitialElement»* or *«FinalElement»* (i.e., the first or last activity); *«Conditional»*, which enables creating disjoint branches of the workflow; and *«Fork»* or *«Join»*, which allow starting and ending parallel branches of the workflow The latter represents an action that should be executed to develop the process and has been grouped into three metaclasses: *«OrchestrationActivity»*, which represents an orchestration activity (i.e., an activity performed by a machine); *«ComplexActivity»*, which allows including a process within another process; and *«HumanActivity»*, which represents an activity that someone performs

4 A Model-Driven Solution to Execute and Orchestrate Processes

This section presents a model-driven solution to support the execution and the orchestration of processes into PLM$_4$BS. For this purpose, our solution is composed of: (*i*) a Process Execution and Orchestration Metamodel (PEOMM); and (*ii*) transformation rules to generate PEOMM from the modeling phase and, later, generate executable code. Both aspects are described in Sects. 4.1 and 4.2, respectively.

4.1 Defining a Metamodel to Support Execution and Orchestration

PEOMM has been defined with more granularity (i.e., lower level of abstraction) than PMM, which was briefly introduced in Sect. 3. PEOMM has the form of a MOF-compliant metamodel and incorporates required attributes normalized according to ISO/IEC TR 24744 [33]. PEOMM aims to represent the process structure from the point of view of its execution context.

For this purpose, it provides a complete and theoretical specification to allow executing process models within BPMS. PEOMM (Fig. 2) also describes static and dynamic semantics of this execution. On the one hand, static semantics refers to: (*i*) static information (i.e., specific properties or attributes) of each concept defined in PEOMM; and (*ii*) semantic constraints to ensure building well-formed models. On the other hand, dynamic semantics refers to: (*i*) what information is generated and managed in a dynamic manner (i.e., at runtime); and (*ii*) how and when each element of PEOMM can be instantiated. This semantics enables each element to react and evolve at runtime along its own lifecycle.

Before going further, it is worth clarifying that the syntax used is not enough to semantically define our metamodel. Consequently, we have used OCL [2] (as recommended by OMG) to add formal constraints, which, in turn, limit possible instantiations and therefore valid process models. All our OCL constraints will not be explained here, since they are out of the scope of this paper and it would become too extensive. Nevertheless, as an illustrative example, just a couple of OCL constraints will be explained in detail.

The main metaclass in PEOMM is the *«ExecutionNodeClass» metaclass,* which represents the executable view of any element in the process. These elements are interrelated to build the process execution flow. Such relationships are modeled by means of the *«ExecutionFlow» metaclass*.

The *«ExecutionNodeClass»* metaclass has two kinds of properties: static properties (i.e., *«name»*, *«description»* and *«isInitial»*) and dynamic properties (i.e., *«status»*). The *«status»* property establishes the lifecycle of each executable element. This lifecycle is composed of five allowed status whose transitions are formalized using a state machine (Fig. 3). Subsequently, each transition is triggered by one unique operation of the *«ExecutionNodeClass»* metaclass. These operations (see Fig. 2) are stereotyped as: (*i*) *«CONTR»* (CONSTructor), which defines the constructor to create a new instance of the *«ExecutionNodeClass» metaclass*; or (*ii*) «*SOP*» (Status OPeration), which identifies operations to update or query the internal status of the executable element.

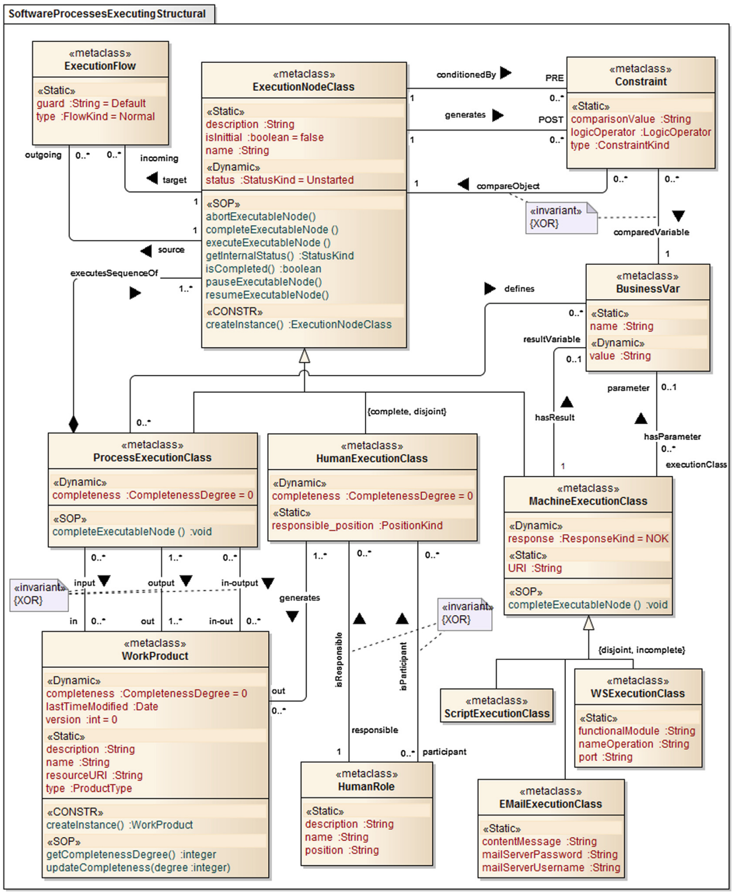

Fig. 2. Metamodel to support the execution and orchestration of processes

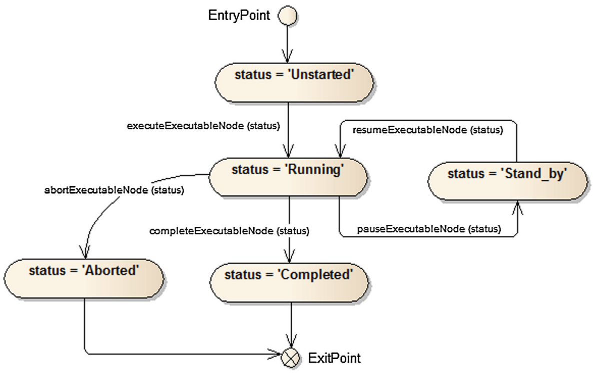
EntryPoint
status = 'Unstarted'
executeExecutableNode (status)
resumeExecutableNode (status)
status = 'Running'
status = 'Stand_by'
abortExecutableNode (status)
completeExecutableNode (status)
pauseExecutableNode (status)
status = 'Aborted'
status = 'Completed'
ExitPoint

Fig. 3. Lifecycle of the *«ExecutionNodeClass» metaclass*

Each operation is formally defined by means of an object-oriented language proposed by UML to define executable semantics [15]. Table 2 shows just a couple of operations since explaining all of them is out of the scope of this paper and it would become too extensive. Operations shown in Table 2 are: *(i) «createInstance»*, which defines how an execution element is instanced; and *(ii) «executeExecutableNode»*, which allows running the execution of an executable element.

Table 2. Some operations of the *«ExecutionNodeClass» metaclass*

```
createInstance (String name,
                String descp, boolean isInittial)  {
     this.name = name;
     this.description = descp;
     this.isInittial = isInittial;
     this.status = "Unstarted";
}
```

```
executeExecutableNode ()  : void   {
     if (this.status.equals ("Unstarted") ||
         this.status.equals ("Stand_by"))
             this.status = "Running";
}
```

The *«ExecutionNodeClass» metaclass* is also specialized in three metaclasses in order to distinguish among different executable elements:

- The *«ProcessExecutionClass»* metaclass. It represents the highest-level execution entity in any process that defines the execution structure of a process. In this sense, the *«ProcessExecutionClass» metaclass* is composed of a set of execution nodes (i.e., *«ExecutionNodeClass»*). This metaclass also includes the *«completeness»* property, which aims to indicate the degree of completeness of the process. This property is also used to establish when the executable process is completed, that is to say, an executable process is completed as follows: *(i)* when its internal status is *«Running»* (this condition is already checked in the *«ProcessExecutionClass»* super-metaclass); and (*ii*) when its degree of completeness is 100%.
- The *«MachineExecutionClass»* metaclass. It is very important because it allows orchestrating and defining the coordination of events among information systems during the process execution. For this purpose, this metaclass includes one static property (*«URI»*, which is the uniform resource identifier of the target system) and one dynamic property (*«response»*, which stores the response code after performing the orchestration activity). Moreover, the *«MachineExecutionClass»* metaclass has been also specialized in three metaclasses in order to distinguish among different automatic events: *«ScriptExecutionClass»*, which allows executing script code; *«WSExecutionClass»*, which makes it possible to invoke Web services; and *«EMailExecutionClass»*, which allows defining notification via email.
- The *«HumanExecutionClass»* metaclass. It represents any executable element that must be performed by human agents. These agents are modeled into PEOMM with

the *«HumanRole»* metaclass, which represents an actor who can be participant or responsible, but cannot perform both roles (PEOMM shows this semantic constraint using the *«XOR»* logical operator between the *«isParticipant»* and *«isResponsible»* associations). Both roles have been considered because many companies (e.g., software companies) have many work teams where each member cooperates in developing activities depending on the involvement degree. The difference among these roles is that participants can complete some aspects of the product, but they cannot complete the activity (this action is only available for responsibles).

So far, this paper has explained how PEOMM defines the internal semantics of each *«ExecutionNodeClass» metaclass* using a status machine whose transitions are triggered by operations. Nonetheless, it is also important to establish what conditions have to be true before executing each *«ExecutionNodeClass» metaclass* and after running them. The former are named pre-conditions. They are used to capture a conjunction of events that lead to the execution of an *«ExecutionNodeClass»* and allows defining conditions such as, «*a specific activity may not be executed until either the previous activity is completed or a specific business rule is true*». The latter are named post-conditions and they allow defining conditions such as, «*after executing the current activity, the output work products must have been completed*».

PEOMM models these pre- and post-conditions with the *«Constraint» metaclass*. This metaclass enables comparing (with logic operators) a specific comparative value either of business variables or the internal status of *«ExecutionNodeClass»*. The concept of business variable is key in PEOMM because it helps store values and results used in orchestration activities. The *«BusinessVar» metaclass* models business variables in PEOMM.

Finally, PEOMM also takes into account the dynamic behavior of the results (*«WorkProduct» metaclass*) of a software process because they may evidence the process completion in, e.g., audits. In addition, the products evolve during the process execution, (i.e., its version or finishing percentage, for instance).

After introducing the previous metaclasses, Table 3 describes one of the most important OCL constraints to build well-formed execution models. This constraint is defined at the *«ProcessExecutionClass» metaclass* and checks three conditions: (*i*) a process cannot contain itself, in order to avoid an indefinite execution model because of recursive definitions; (*ii*) each executable process can only contain one executable node typed as initial; and (*iii*) each work product should have been generated by an instance of the *«HumanExecutionClass» metaclass,* which should also belong to the *«ProcessExecutionClass» metaclass.*

4.2 Defining a Transformation Protocol

The architecture of PLM$_4$BS (Sect. 3.1) considers the use of MDE to obtain the PEOMM and its executable version of the process from PMM. For this purpose, PLM$_4$BS defines a transformation protocol based on three steps:

1. **Generating systematically the basic Process Execution and Orchestration Model (PEOM).** The basic PEOM is considered the first version of PEOM and it is systematically obtained from the process modeling model using a comprehensive

Table 3. OCL constraint of the *«ProcessExecutionClass» metaclass*

```
context ProcessExecutionClass inv:
  let activitiesColl : SortedSet =
    self.ExecutionNodeClass→select(oclIsTypeOf(HumanExecutionClass))
  in
  self.ExecutionNodeClass→
       select(oclIsTypeOf(ProcessExecutionClass))→count(self)=0
  and
  self.ExecutionNodeClass→
       iterate(node: ExecutionNodeClass; acc:Integer=0 |
               if node.isInittial then acc + 1 else acc endif)=1
  and
  self.out→forAll (p1 : WorkProduct|activitiesColl→
                    exist(a:HumanExecutionClass |
                          a.WorkProduct→exist(p2:Product|p1=p2)))
  and
  self.out→size() = activitiesColl→
       iterate(a : HumanExecutionClass;
               acc : Integer=0 | acc+ a.Product→size());
```

set of M2 M transformation rules. Table 4 describes the *«toHumanExecution Class»* rule using QVT (the others can be found in [34]). This rule describes how *«HumanExecutionClass»* (PEOMM) is obtained from *«HumanActivity»* (PMM). Firstly, this rule initializes all static and dynamic properties (line 3 and 4) of the *«HumanExecutionClass» metaclass* and resolves all relationships with *«WorkProduct»* and *«HumanRole» metaclasses* (lines 6–9). This QVT rule also uses some auxiliary functions: *«isInittialActivity»*, which checks if the *«HumanActivity» metaclass* is the first metaclass in the process; *«createPreConditions»* and *«createPosConditions»*, which elaborate the constraints associated with the *«HumanExecutionClass»*.

2. **Generating manually the final PEOM.** Once the previous step is carried out, the process engineer can add his/her knowledge to the basic PEOM in order to complete the execution context. This unsystematic and manual transformation generates the final version of PEOM. At this point, it is important to highlight that, if the process engineer detects deficiencies in the structure of the execution model, these changes must be extended to the process modeling model in order to avoid inconsistency among models. This procedure can be indefinitely repeated in order to achieve a coherent execution and organization model.
3. **Generating executable code.** Finally, PLM$_4$BS defines a comprehensive set of M2T transformation rules to obtain executable code from the final PEOM. These rules have been defined in MOFM2T and allow generating WS-BPEL code that can be executed in most BMPS [11]. Table 5 describes the *«defineBPELStructure»* rule using MOFM2T (the others can be found in [17]). This rule describes how the WS-BPEL structure of the process is obtained from PEOM.

Table 4. QVT rule to obtain the «HumanExecutionClass» metaclass

```
mapping HumanActivity::toHumanExecutionClass () :
        SPExecutionMetamodel::HumanExecutionClass    {
  description := self.description;       status := "Unstarted";
  isInittial  := isInittialActivity ();  name   := self.name;
  responsible_position := self. responsible_position;
  generates += self.Product→forAll (p : Product |
               p.resolveone(WorkProduct);
  responsible := self.isResponsible.resolveone(HumanRole);
  participant += self.isParticipant→forAll (rol : Stakeholder |
                 rol.resolveone(HumanRole);
  PRE  := createPreConditions (); POST := createPosConditions ();
}
```

Table 5. MOFM2T rule to generate WS-BPEL structure of the process

```
[template public defineBPELStructure (process : ProcessExecutionClass)]
  [for (node : ExecutionNodeClass | process.ExecutionNodeClass)]
    [if (node.oclIsTypeOf (HumanExecutionClass)]
       <extensionActivity>
       <b4p:peopleActivity name="[node.name/]"
                           inputVariable ="ToBeSpecified"
                           outputVariable="ToBeSpecified">
       <b4p:localTask reference="[ node.name /]_TSK" />
       </b4p:peopleActivity>
       </extensionActivity>
    [elseif (node.oclIsTypeOf (WSExecutionClass)]
       <invoke name="[ node.name /]" partnerLink="[ node.name /]_PL"
           portType="[node.port/]" operation="[node.nameOperation/]"
           inputVariable="[ node.hasParameter.name /]"
           outputVariable="[node.hasResult.name/]"createInstance="yes"/>
    [elseif (node.oclIsTypeOf (EMailExecutionClass)]
       <invoke name="[ node.name /]" partnerLink="[ node.name /]_PL"
           portType="ToBeSpecified"  operation="ToBeSpecified"
           inputVariable="[ node.hasParameter.name /]"
           outputVariable="[node.hasResult.name/]"createInstance="yes"/>
    [elseif (oclIsTypeOf (ProcessExecutionClass)]
       <!-- … -->
    [else] <!-- ScriptExecutionClass -->
       <empty name="[ node.name /]" />
    [/if]
  [/for]
[/template]
```

5 Discussion, Future Work and Conclusions

In recent years, standards and guidelines (such as PMBOK, PRINCE2, CMMI or ISO 9001, among others) recommend that organizations should formally manage their processes in order to achieve lower costs and improve quality and productivity.

To meet these goals, companies should carry out an effective BPM of their processes to achieve the continuous improvement of such processes.

However, in the context of software organizations, applying BPM is not a simple task due to features of software processes. This situation provokes that software companies usually focus on defining their processes although, later, execution and orchestration are manually and/or unilaterally performed by each involved role. Consequently, software process becomes difficult to execute, manage, maintain, monitor, evolve and measure.

At present, there are many PMLs [18], but just a few of them include mechanisms for supporting the execution of the process. In addition, none of them is mature enough to comply with the commitment pursued. SPEM2.0 standard could be the solution, but its complexity and non-executability makes it impossible. Regarding executability, each proposal presented in Sect. 2 offers mechanisms to perform the process into ad hoc systems, what make harder its application in real environments because companies already use specific process engines. It is interesting to underline that none of these proposals mention mechanisms to support the orchestration of processes.

This paper proposes a MDE-based solution to execute and orchestrate the software process in real environments since it is oriented to be applied, in an integrated way, to enterprise management systems (such as BPM suite [11]). In fact, some papers have been published to report successful cases [12, 14].

Finally, the publication of this paper opens new and interesting future lines of work. On the one hand, we plan to support monitoring and continuous improvement phases of the architecture of PLM$_4$BS in order to assess the execution of software processes. On the other hand, we aim to research how simulation mechanisms can be included in PLM$_4$BS so as to support decision-making procedures related to resource allocation or deadlock identification, among other aspects.

Acknowledgments. This research has been supported by the POLOLAS project (TIN2016-76956-C3-2-R) and by the SoftPLM Network (TIN2015-71938-REDT) of the Spanish the Ministry of Economy and Competitiveness.

References

1. Van-der-Aalst, W.M.P.: Business process management: a personal view. Bus. Process Manage. J. **10**(2), 5 (2004)
2. ISO/IEC. ISO/IEC 19507:2012 Information technology, Object Constraint Language (OCL). International Organization for Standardization, formal/2012-05-09 (2012)
3. Ruiz-González, F., Canfora, G.: Software process: characteristics, technology and environments. SPT Softw. Process Technol. **5**, 5–10 (2004)
4. Piattini-Velthuis, M., Ruiz-González, F., Canfora, G.: Software process: characteristics, technology and environments. SPT Softw. Process Technol. **5**, 5–10 (2004)
5. Papazoglou, M., Ribbers, P.: E-Business: Organizational and Technical Foundations. Wiley, New York (2006). ISBN-13: 978-0470843765
6. Pedraza, G., Estublier, J.: Distributed orchestration versus choreography: the FOCAS approach. In: Wang, Q., Garousi, V., Madachy, R., Pfahl, D. (eds.) ICSP 2009. LNCS, vol. 5543, pp. 75–86. Springer, Heidelberg (2009). doi:10.1007/978-3-642-01680-6_9

7. García-García, J.A., Alba, M., Escalona, M.J.: Software Process Management: A Model-Based Approach. Information Systems Development: Building Sustainable Information Systems, pp. 167–178. ISBN: 978-1-4614-7539-2 (2013)
8. Schmidt, D.C.: Model-driven engineering. IEEE Comput. **39**(2), 25–31 (2006). IEEE Computer Society
9. OASIS. Web Services Business Process Execution Language. Organization for the Advancement of Structured Information Standards (2007). http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html
10. OMG. XMI BPMN, Business Process Modeling Notation (2011). http://www.omg.org/spec/BPMN/2.0/
11. Meidan, A., García-García, J.A., Escalona, M.J., Ramos, I.: A survey on business processes management suites. Comput. Stand. Interfaces (2017). doi:10.1016/j.csi.2016.06.003
12. Garcia-Garcia, J.A., Enriquez, J.G., Garcia-Borgoñon, L., Arevalo, C., Morillo, E.: A MDE-based framework to improve the process management: the EMPOWER project. In: IEEE 15th International Conference of Industrial Informatics (2017)
13. Salido, A., García-García, J.A., Ponce, J., Gutiérrez, J.: Tests management in CALIPSOneo: a MDE solution. J. Softw. Eng. Appl. (2014). doi:10.4236/jsea.2014.76047
14. García García, J.A., Escalona, M.J., Martínez-García, A., Parra, C., Wojdyński, T.: Clinical process management: a model-driven & tool-based proposal. In: Information Systems Development: Transforming Healthcare through Information Systems. ISBN: 978-962-442-393-8 (2015)
15. OMG. Semantics of a Foundational Subset for Executable UML Models v1.1. Object Management Group (2013). http://www.omg.org/spec/FUML/1.1/
16. García-García, J.A., Escalona, M.J., Domínguez-Mayo, F.J., Salido, A.: Methodological tool solution in the model-driven engineering paradigm. J. Softw. Eng. Appl. (2014). doi:10.4236/jsea.2014.74022
17. Martínez-García, A., García-García, J.A., Escalona, M.J., Parra, C.L.: Working with the HL7 metamodel in a Model Driven Engineering context. J. Biomed. Inf. (2015). doi:10.1016/j.jbi.2015.09.001
18. García-Borgoñón, L., Barcelona, M.A., García-García, J.A., Alba, M., Escalona, M.J.: Software process modeling languages: a systematic literature review. Inf. Softw. Technol. **56**, 103–116 (2014)
19. Bendraou, R., Sadovykh, A., Gervais, M.P., Blanc, X.: Software process modeling and execution: the UML4SPM to WS-BPEL approach. In: 33rd Conference on Software Engineering and Advanced Applications, pp. 314–321. ISBN: 0-7695-2977-1 (2007)
20. Ferreira, A.L., et al.: An approach software process design and implementation using transition rules. In: Software Engineering and Advanced Applications Conference (2011)
21. Wise, A.: Little-JIL 1.5 Language Report. Department of Computer Science, University of Massachusetts, Amherst, MA, UM-CS-2006-51 (2006)
22. Di Nitto, E., Lavazza, L., Schiavoni, M., Tracanella, E., Trombetta, M.: Deriving executable process descriptions from UML. In: Proceedings of the 24th International Conference on Software Engineering ICSE, pp. 155–165 (2002)
23. Cugola, G., Di Nitto, E., Fuggetta, A.: JEDI event based infrastructure and its application to development of OPSS WFMS. Trans. Softw. Eng. **27**(9), 827–850 (2001)
24. Chou, S.C.: A process modeling language consisting high level UML-based diagrams and low level process language. J. Object Technol. **1**(4), 137–163 (2002)
25. OMG. SPEM, Software & Systems Process Engineering Metamodel specification. Object Management Group (2008). http://www.omg.org/spec/SPEM/
26. Bendraou, R., Combemale, B., Cregut, X.: Definition of an executable SPEM 2.0. In: 14th Asia-Pacific Software Engineering Conference, APSEC 2007. IEEE, pp. 390–397 (2007)

27. Havey, M.: Essential Business Process Modelling. ISBN-13: 978-0596008437 (2005)
28. Hill, J.B., et al.: Gartner's Position on Business Process Management. Business Issues. Gartner, Stamford (2006)
29. Escalona, M.J., Gutierrez, J., et al.: Practical experiences in web engineering. In: Advances in Information Systems. Advances in Information Systems Development (2007)
30. Bosch, J.: From Software Product Lines to Software Ecosystems, pp. 111–119 (2009)
31. OMG. Query/View/Transformation (2017). http://www.omg.org/spec/QVT/1.0/
32. OMG.MOF Model to Text Transformation Language (MOFM2T) (2017). http://www.omg.org/spec/MOFM2T/1.0/
33. ISO/IEC. ISO/IEC TR 24744:2007 Software and systems engineering Lifecycle management Guidelines for process description (2007)
34. García-García, J.A.: A proposal for the use of the model-driven paradigm (MDE) for the definition and execution of business processes. Ph.D. thesis (2015). https://documat.unirioja.es/servlet/autor?codigo=3722430