|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  | | --- | | Automatic Deployment and Monitoring of Software Processes | | Marília Aranha Freire1,2, Fellipe Araújo Aleixo1,2, Uirá Kulesza1 | | 1Federal University of Rio Grande do Norte (UFRN), Natal, Brazil  [uira@dimap.ufrn.br](mailto:uira@dimap.ufrn.br) | | 2Federal Institute of Education, Science and Technology of Rio Grande do Norte (IFRN), Natal, Brazil  [marilia.freire@ifrn.edu.br](mailto:marilia.freire@ifrn.edu.br), [fellipe.aleixo@ifrn.edu.br](mailto:fellipe.aleixo@ifrn.edu.br) | |

Keywords: Software Process, Software Process Lines, Metrics.

Abstract: This paper presents an approach based in model-driven strategies and techniques to automatically deploying and Monitoring Software Processes. It consists of two stages: (i) modelling and integration of metrics in software process and; (ii) automatic collection of metrics during the execution of the process as a workflow. In order to evaluate the feasibility of our approach, we have designed and implemented it using existing and

# 1 INTRODUCTION

[Motivar a importancia de metricas, mas ressaltar que existe pouca preocupacao e trabalhos explorando o deployment automatico de metricas em ambientes de execução de processos... (embora muitas sao usadas em cenarios de processos de negocio)...]

...

The remainder of this paper is organized as follows. Section 2 presents existing model-driven approach to managing variabilities and gives an overview of model-driven strategies and techniques to automatically monitoring software processes proposed in this paper. Section 3 describes the implementation issues as composing metrics in software processes and automatic deployment of software processes and metrics using existing model-driven technologies. Some related works are presented in the Section 4. Finally, Section 5 presents the conclusions and points out future work directions.

# 2 Automatic monitoring of software development processes

2.1 A Model-Driven Approach to Managing Variabilities

This work is based in an approach defined by Aleixo et al. [REF -ICEIS]. The overview of the initial definition of the approach could be seen in . The main goal of this approach is to allow the derivation of specific software development process specification according with the Eclipse Process Framework – EPF. This software development process fits the specific needs of a software development project. The approach applies the software product line concepts [REF] to software development process, reinforcing the premise that software processes are software too [REF].

As shown in , the original approach could be defined in three stages: (i) process definition, (ii) process derivation, and (iii) process deployment and execution. In the initial stage a specific software process family is selected, as a set of software process that shares the same structure and some common process elements [REF]. After the analysis of each member of the software process family, identifying the commonalities and variabilities among the individuals, an EPF process specification is created and the variation points are annotated in the specification. These annotations could be recognized by a product derivation tool, adapted to this purpose, and interpreted to generate the models necessary to the product derivation: (i) process model, (ii) variability model, and (iii) configuration model [REF].

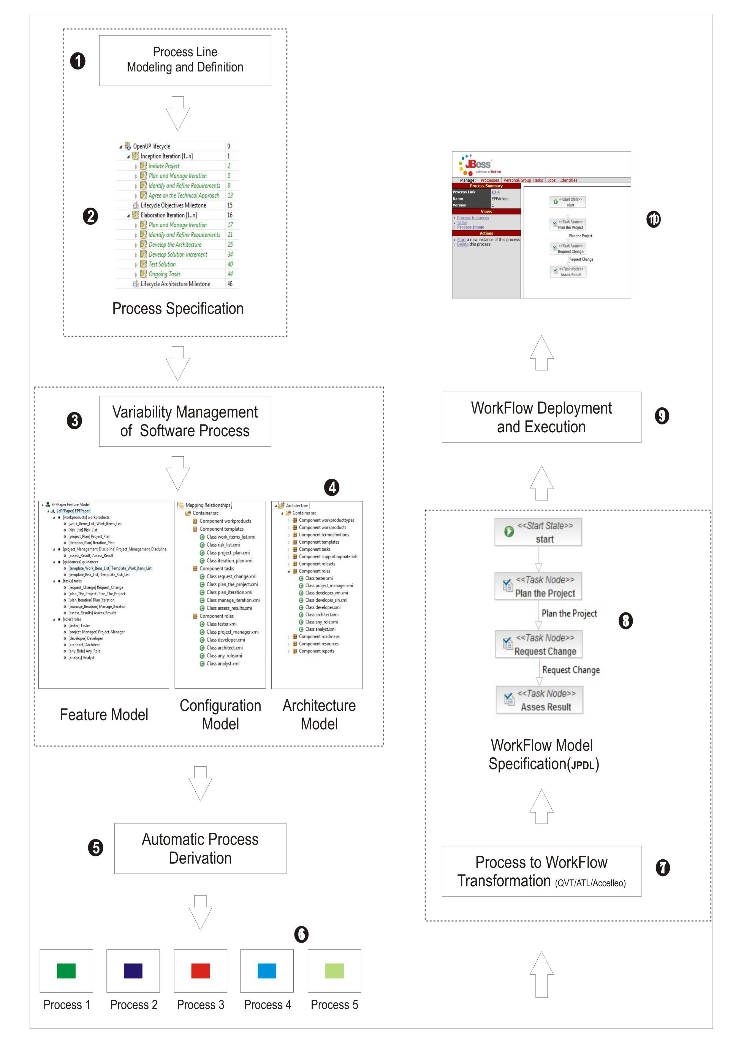
The second stage is intended to the software process derivation. The execution of this stage is motivated by a new software development project. After the analysis of the particularities for this new software process, a feature selection is done in an instance of the variability model. With this feature selection, a product derivation tool could be used to derivate such specific process. The result of this derivation is an EPF process specification according to the selected features.

Figure : Based Approach

In the third stage, the approach provides the deployment and execution of the software process specification with the support of a workflow engine. To able this intent, the process specification have to be transformed in a workflow specification. The transformation between these two models is possible with the support of model-to-model transformation language as operational QVT [REF]. To able the generation of the deployment files, beyond the workflow specification are needed some well-known information about the future execution of the software process, information such as: (i) number of iterations in each phase (assuming that the process is based in the Unified Process – UP), (ii) people playing specific roles, (iii) milestones dates, (iv) release dates, etc. The workflow specification and the well-known execution information are used in a model to text transformation to generate the workflow deployment files. These generated files are likely to be deployed in a workflow engine.

After the deployment in the workflow engine, a new instance of this process could be started. The process execution could be monitored through the workflow engine console, which coordinates the interaction of the process stakeholders and the process instance execution. The workflow engine console indicates which the actual activity that has to be done in the process execution, and also could store some information about the execution, such as: (i) start and end date of an activity, (ii) some related observations, (iii) produced artifacts, etc.

2.2 Model-Driven Strategies and Techniques to Automatically Monitoring Software Processes

In this section, we present an overview of our approach that integrates software metrics in a process definition, customization and execution. It is based on the approach outlined in Section 2.1 of this document. illustrates by rectangles with dashed borders the main elements of the base approach that have been adapted in this work. This tailored approach combines a metric model with a process model to enable the automatic metrics collection during the execution of a software process.

Initially we have to specify phase like the original one. In this phase we have two new steps: the metrics specification and the metrics selection. The first step refers to metrics addition in the software process. It is realized during the process definition using tools such as EPF, as the based approach. After that, the second new step is concerned to the metrics selection that we can get from the process feature model that now includes options concerning the defined metrics.

On the second stage we need specify the composition of metrics as a metric model. This modeling is made specifying for each metric selected which process tasks need to be intercepted to collect data automatically during the process execution. In the end of this phase we have two independents models: process and metrics.

Finally, the third stage is the automatic transformation from the software process specification to a workflow specification. Here, we integrate metrics actions into the execution of the base process tasks performing a sort of *weaving*. Through these transformations, the sequence of process activities and the process metrics are mapped to a workflow definition. This step makes possible the deployment and execution of automatic metrics collection during the process execution in a workflow engine.

In order to evaluate the feasibility of our approach, we have designed and implemented it using the same based approach technologies. The next section describes the implementation of this new approach detailing an example of metrics integrated into OpenUp software process.

# 3 Implementation Issues (melhorar esse nome)

3.1 Automatic Derivation of Software Processes

(Figura com modelos de variabilidade, e sua respectiva selecao, dando ideia de um exemplo de processo que será explorado)

The starting point for implementing this extended approach is inclusion of metrics in the process definition. This inclusion aims to define activities relating to the metrics collection within the process, as well as analyzing and defining some types of metrics used in software development projects. At this moment, this approach is limited to measurement the schedule durations of significant project tasks.

To enable the inclusion of metrics in customized OpenUP process is necessary to include this optional activity in the specification of OpenUP process family. As described in Section 2.1, after define the process line comes the variability management phase where now besides be able to select features that represent process activities, can be selected also activities relating to the collect of metrics that will be held during project execution. The feature model now includes the metrics selection as optional and alternatives features. Figure X shows a feature model containing metrics selection defined for OpenUp process with some metrics selected.

After the metrics selection, these metrics have to be intercalated with the process tasks. The next section describes how this structure is implemented.

3.2 Composing Metrics in Software Processes

This section presents the metamodel for composing metrics in software processes, it allows to describe where the metric collect can be plugged into the workflow that represents the process execution.

It supports activities and tasks like intervention points for the metrics collection. Therefore, it is necessary to specify for each metric which activities or tasks it must operate through a simplified model of metrics.

shows the metamodel for the composition of metric and process. It describes only the main information for the composition, realized by a model to model transformation. This modeling looks like a metrics plan.

The *MetricModelPlan* element must aggregate the metrics set defined for the process. It has the attribute *name* that must define the metric plan name. The central model element is the *Metric*. It must be used to describe all metrics defined for the process. Each metric modelled has a unique identifier named *id*. The attribute *name* contains the metric name. And in *description* attribute must be specified a description of metric.

The parameter *type* contains the metric type that can be one of the following: *hardData*, *softData* and *normalizedData*, like specified in the enumeration named *MetricType*. This classification identifies the kind of information used in measurement [REF]. The hard data information can be quantified with little or no subjectivity. The soft data information refers the kind of information which human opinions must be evaluated or, in other words, information where precision is not possible. Finally, the normalized data is used to evaluate whether projects are above or below normal in terms of productivity or quality. They represent standard metrics for comparative purpose. This work focus on the hard data measurements initially.

The *form* property represents the type of time collection for a metric. The *continuous ColectType* means that the task can´t be paused and the time spent is the time from start until the end of task or activity and the *intercalated* *ColectType* means that the task can be paused and the final time is the sum of each timeslice spends on it.

At end, the *unit* property must be used for adding other simple information for the metric definition: the metric´s unit, it can be choose in the *MetricUnit* enumeration. Initially we have only two options: hours or *UC*.

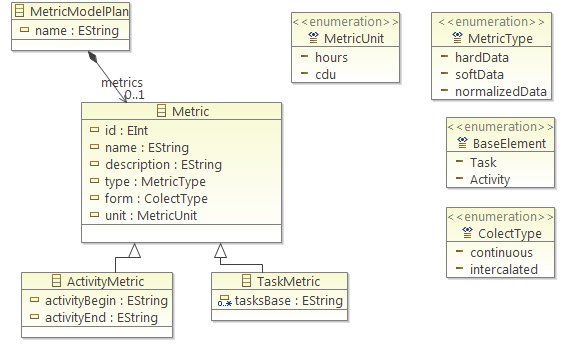


Figure : The Metric Metamodel Composition

While modelling a metric plan, we have to choose a base element for the metric intercept. This processes elements are activities or tasks. So, we have two metric specializations: *ActivityMetric* and *TaskMetric*. The *ActivityMetric* element must be used to define a metric which must intercept processes elements of activity type. This element adds two attributes for the metric. The attributes “*activityBegin*” and “*activityEnd*” that store the first and final activity name that this metrics will intercept, respectively, representing a sequence of tasks to be monitored. If the metric is to intercept only one activity, the *activityEnd* attribute must be blank. If there isn´t a flow between the activities only these two activities will be considered. And the *TaskMetric* element adds only one new attribute named *tasksBase* that must contain the names of all tasks affected by the metric being defined.

At end, the property variables can be used for adding other important information for the process execution but it still is not being used.

The second element in illustrates a metric model example for an *OpenUp* process composition where is specified two metrics for activities interceptions *Duration of all UC development tasks* and *Duration of UC requirements*. The metric *Duration of all UC development tasks* for example is defined to intercept the process activities to count time spent on the process with activities related to developing a use case (UC). The model exemplified sets activities from “develop\_solution” activity to “test\_solution” activity. It means that all activities between them have to be monitored. The value *continuous* setting to the form attribute implies that the time measured by this metric is the sum of all the time spent for each activity specified.

3.3 Automatic Deployment of Software Processes and Metrics

In order to define the scope of our case study, initially were extracted from the OpenUp specification the different elements of elaboration phase, focusing its activities and tasks. Figure 3 shows an implementation view of the approach to fragments of the case study. As can be seen, the approach is performed in three steps. The stage A illustrates the case study in the form of an element structure (Work Breakdown Structure) of EPF and the metrics model defined. Steps B and C illustrate resulting artifacts of the M2M and M2T transformations, respectively.

1. Model to Model Transformation

The M2M transformation is responsible for realize the process and metrics composition. In this transformation all activities or tasks specified in the metric model must derivate new elements that will be responsible for the automatic metric collect.

Table : Mappings for JPDL Elements

|  |  |
| --- | --- |
| UMA or Metric Element | JPDL Element |
| Activity | task-node |
| Worker-order | transition |
| Activity without predecessor | start-node |
| Activity without successor | end-node |
| Activities with more than one successor | fork-node |
| Activities with more than one predecessor | join-node |
| Task | task |
| Metric | event and action |

So, the metrics and software process integration is made at the moment of workflow model generation, when the model to model transformation is performed (UMA to JPDL transformation). The Step 1 of Figure X illustrates a fragment of *OpenUp* elaboration phase. It show some activities like *Plan and Manage Iterations*, *Develop Solution Increment*, and expand some activity showing their tasks like for example *Design Solution*, *Implement Developer Tests* and others regarding their respective activities parents.

Each metric is associated with JPDL actions elements that are triggered by JPDL events elements in the end of each task execution. These actions elements are responsible for the data collection through its association with Java source code, how can be visualized in Step C of Figure X.

The Table x shows the origin of each JPDL element produced by M2M transformation.

So, this integration represents a transversal composition with the original UMA process. This composition is performed with QVTO transformation language which receives a UMA process model and a metric model as input and produces an JPDL process execution model (workflow model) as output. The metric and process input models are presented in the Stage A of Figure X, as well as the JPDL source code output in Step B. At the end of this stage, we have a workflow model that is the weaving of the process and metrics models. We must consider that the elements which enable the measurements (actions and events) aren't mapped visually in the workflow definition but they are added as technical details are highlighted with dashed rectangles in the source code JPDL.

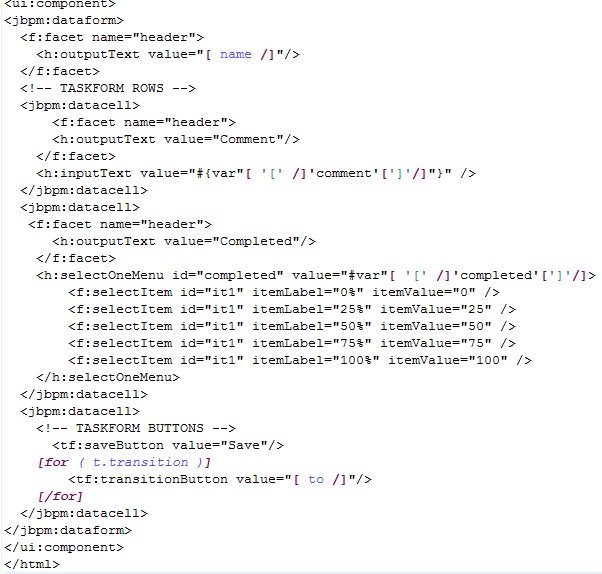


Figure : Acceleo Template Example

1. Model to Text Transformation

As the original approach, for the text to model transformation was used Acceleo, which is a model generation language OMG standard, based on an approach to DDM. Model to text transformation uses as input the model jPDL resulting from transformation from model to model, described in section earlier.

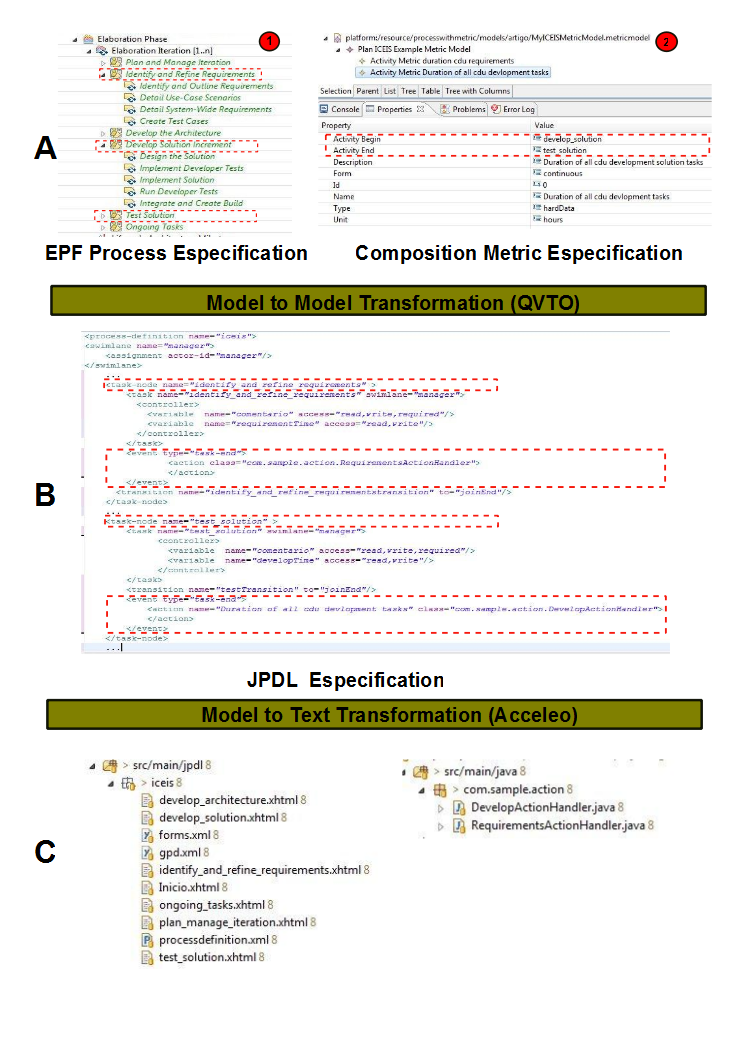
The jBPM enables the creation of web forms implemented in Java Server Faces (JSF) framework, from a JPDL workflow model definition. Such forms can be used to track the flow of process execution. This possibility of monitoring is responsible for storing information about the tasks and /or decisions taken during its execution.

Figure : Approach Results

Figure X shows an Acceleo template for composition of the JSF pages that will complement with information the tasks of our workflow. Some jBPM taglibs were used for the engine to run the form and associate it with the process. We can see the template configuration information for the tasks and transitions between them. The transformation model-to-text Acceleo are also responsible for generating a configuration file (forms.xml) that lists each task of our flow to a JSF form as well as java files that represent the custom actions in accordance with the metrics used. All these files were generated on a project in Eclipse jPDL. Through simple manual settings jBPM engine, the process can easily be deployed and run on it. Stage C of Figure X shows the files resulting from the processing M2T.

3.4 Monitoring of Software Processes with Metrics

Figure x shows a view of the process running in the browser via the jBPM engine. After the deployment of the workflow process in jBPM, the user can run a new process instance, where he can view the current state of the process, which presents the details of those tasks performed as well as the current values of process variables. After performing each task, the user informs in the workflow execution interface, some information (such as date/time of onset, date/time of completion, current status, artifacts implemented) to monitor the activity in question. All information is stored in a database, and related process instance. All these steps are repeated for each task by the end of the process when the final state of the workflow has been reached.

[Falta criar uma figura da execução e explicar um pequeno exemplo colhendo o tempo para as atividades interceptadas pelas medidas]

**4 Related Work**

- algumas abordagens propostas promovendo a execucao de processos, mas nenhuma com foco no deployment de metricas para fins de monitoramento do processo

- trablahos de processos de negocio

**5. Conclusions**

**ACKNOWLEDGEMENTS**

This work was supported partially by Brazilian Research Council (CNPq) under grants: No. 313064/2009-1, No. 552010/2009-0, and No. 480978/2008-5.

**REFERENCES**