|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  | | --- | | 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

...

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 presenteds in the section 4. Finally, Section 5 presents the conclusions and points out future work directions.

[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)...]

# 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 Figure X. 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 Figure X, 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.

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

[- Mostrar em linhas gerais como estamos adaptando o processo anterior para monitorar processos de software. olhar na figura do ICEIS/SBES/SBCARs, e ver quem os quadrados tracejados que vao entrar para poder descreve-los em linhas gerais um-a-um)]

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. Next we briefly explain the stages of the proposed approach.

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 possible 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 process execution. In the end of this phase we have two independents models: process and metrics.

Finally, the third stage is the automatic transformation of 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 activities of the process 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. also provides an overview of the implementation of our approach.

Section 4 describes this new approach in action by detailing an example of metrics integrated into the software process.

This work promotes the automatic metrics collect..

Falar dos benfícios da abordagem????

# 3 Implementation Issues (melhorar esse nome)

3.1 Automatic Derivation of Software Processes

(mostrar uma figura de um processo com os modelos de variabilidade, e sua respectiva selecao, dando ideia de um exemplo de processo que será explorado - focalizar fluxo de requisito e implementacao)

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 in analyzing and defining the most common types of metrics used in software development projects. At this moment, this approach is limited to measurement the schedule durations of significant project tasks.

This alter was implemented using the EPF Composer, only inserting new process elements in a traditional software process definition.

As described in section 2, 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.

After the metric inclusion, the process execution must perform the automatic metric collection. These metrics have to be intercalated with the process tasks execution.

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 a 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 metric and process composition. It describes only the main information for the transformation composition. The modelling creation looks like a metrics plan.

The element *MetricModelPlan* aggregates the set of metrics defined for the process. It has the attribute *name* that must define the metric plan name. The main 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. In attribute *description* must be specified the metric´s description.

The parameter “*type*” contains the metric type that can be one of the following: *hard data*, *soft data* and *normalized data*, like specified in the enumeration named *MetricType*. This classification adopts the taxonomy presented in [] where ... Firstly, the hard data software is our interest.

The property “*form*” represents the type of task time collection. The *ColectType* *continuous* means that the task can´t be paused and the time counted is the time from start until the end of task and the *ColectType* *intercalated* means that the task can be paused and the final time is the sum of each timeslice spend on it. This information will be useful in future when we are generating the process workflow.

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

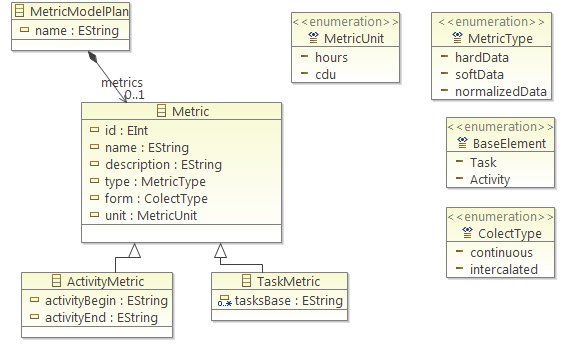


Figure : The Metric Metamodel Composition

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. 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 illustrates a metric model example for an *OpenUp* process composition where is specified two metrics for activities interceptions. The metric “*Duration of all cdu development tasks*” is defined for intercept the process activities from “develop\_solution” activity to “test\_solution” activity. It means that all activities between them have to be monitored.

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. illustrates a fragment of the OpenUP process specification using the Eclipse Process Framework (EPF). It shows some of elaboration phase activities and tasks.

1. Model to Model Transformation
2. Model to Text Transformation

(aproveitar exemplo ou descricao/texto do artigo SBCARS)

- Mapeamento Modelo-para-Modelo

- Mapeamento Modelo-para-Texto

This automatic deployment and monitoring approach has two model to model (m2m) transformations. The first transformation represents the process base customization to include new tasks that will be responsible for the metrics collections. This stage is useful for the process users viewing process points where metrics are collected. The second m2m transformation is responsible for realize the composition. In this transformation, all activities or tasks specified by the metric model must derivate new elements that represent events and associated actions. These actions are responsible for performed the metric calculation.

So, the metrics and software process integration is made at the moment of workflow model generation, when the model to model transformation is performed, the UMA to JPDL transformation.

After the intercept points of the metrics are defined in the metrics model, all tasks related by metrics are associated with actions that are triggered by events at the beginning and end of the each task/activity execution. This integration represents a transversal composition with the original approach. This composition is performed with QVTO transformation language which receives a process model and a metric model as input and produces an execution process model (workflow model) as output. The metric metamodel of input was presented in the previous section and the process metamodel is UMA. For the output, the workflow metamodel is the JPDL, following the same base approach technology.

At the end of this stage, we have a workflow model that is a weaving of the process and metrics models. We must consider that the elements which represent the measurements aren't mapped visually in the workflow definition but they are added as technical detail.

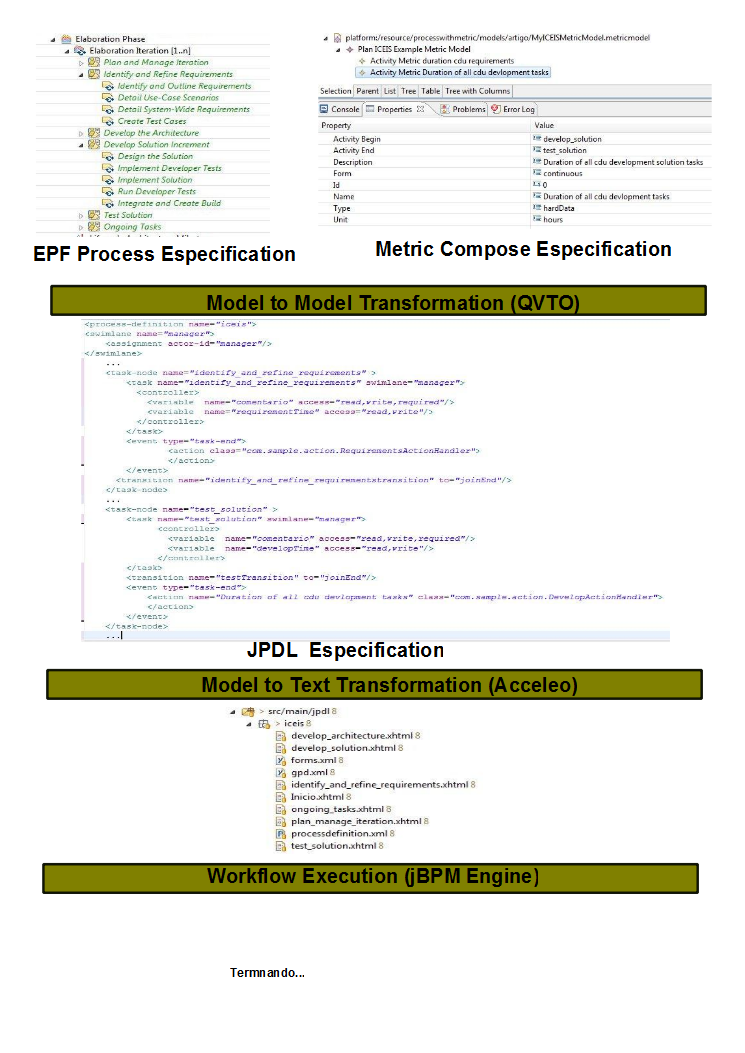
As the original approach is still needed the model to text transformation which is responsible for configuration and the user interface files generation allowing the workflow execution in the jBPM engine.

At this point it is important to supply certain information necessary to deploy the workflow, such as the number of iterations, the roles and actors, as well as the specialization of the metric collect action code when necessary.

3.4 Monitoring of Software Processes with Metrics

The figure x shows the artifacts generated by the approach for an OpenUp process.

**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**