Using the SPEM 2.0 kind-based extension mechanism to define the SPEM4MDE metamodel

Samba Diaw, Mamadou Lakhassane Cisse, Alassane Bah

Using the SPEM 2.0 kind-based extension mechanism to define the SPEM4MDE metamodel

Samba Diaw
Polytechnic Institute (ESP)@Cheikh Anta Diop University, UMMISCO Laboratory
DAKAR
Senegal
samba.diaw@ucad.edu.sn

Mamadou L. Cisse
Polytechnic Institute (ESP)@Cheikh Anta Diop University, UMMISCO Laboratory
DAKAR
Senegal
lakhassane.cisse@ucad.edu.sn

Alassane Bah
Polytechnic Institute (ESP)@Cheikh Anta Diop University, UMMISCO Laboratory
DAKAR
Senegal
alassane.bah@ucad.edu.sn

ABSTRACT

The objective [1] of the OMG's standard SPEM is to propose shared concepts for describing software and even systems processes. The SPEM 2.0 metamodel proposes concepts that are quite generic to describe model-driven development processes. Indeed, the artifacts of those processes are essentially models and relationships between them are numerous (e.g. impact, matching, overlap and so on). We notice that is difficult to have a process modeling language that is suitable to define any kind of process including MDE ones. To overcome this lack, we propose in this paper an extension of SPEM4MDE based on the SPEM 2.0 kind-based extension mechanism. It allows process designer to refine SPEM concepts in order to define the model- driven processes. To illustrate our approach, the MDE-based VUML process for models composition has been used.

KEYWORDS

Model-Driven Engineering (MDE), Model Transformations, Process Modeling Language (PML), VUML (View based Unified Modeling Language

https://doi.org/10.1145/3129186.3129199

1 INTRODUCTION

MDE (Model-Driven Engineering) [4, 10] is a software engineering discipline that advocates the use of models and transformations in the heart of software development. The term MDE was proposed first by Kent in [16] and is derived from the OMG's Model Driven Architecture (MDA) initiative [5, 22].

With the emergence of MDE, many organizations have been starting to transform their traditional software development processes into model-driven ones [19].

Kleppe et al. define a model-driven software process as "*a process of developing software using different models on different levels of abstraction with (automated) transformations between these models*" [17]. Therefore, the model-driven software development process [9, 19, 28, 30]– called MDE software process – may be seen as a transformation chain, each transformation consuming one or several input models and producing one or several output models.

The description of a software process is called *process model*. It can be expressed through any specific language or notation, which is called Process Modeling Language (PML). A process model can be enacted when a development team follows the process model during the development life cycle. One of the major advantages of software process modeling is to help developers using a unified and consistent terminology in order to communicate around the process. Software process modeling should also make possible understanding, reuse, evolution, management, and standardization of processes [15]. One of the major advantages of software process enactment is to allow best guidance of development, checking of activities' and transformations' constraints, and managing consistency between artifacts/models.

In the literature, most of Process-centered Software Engineering Environments (PSEEs) and processes modelling languages (PMLs) have been proposed but none of them has gained much attention from the software community.

Indeed, PSEES offer means to define, model, analyze, improve, and automate software processes. An important problem encountered in PMLs/PSSEEs is that software development processes are subject to permanent evolution during enactment. Without managing this evolution (i.e. having an approach that

will be able to describe the process in real life), PSEEs are condemned to fail in being adopted in software industry.
For instance, a process designer wants to define a MDE process with a SPEM language that is not suitable to do this. Then, it is important to give to the process designer, a way that helps him to refine the SPEM concepts at modeling time.
To fulfill this objective, we propose in this article an extension of the SPEM4MDE metamodel based on the SPEM 2.0 kind-based extension mechanism. It allows process designer at modeling time to refine SPEM 2.0 concepts in order to define any kind of model-driven processes. To illustrate our approach, the MDE-based VUML process for models composition has been used
The remainder of the paper is organized as follows: section 2 introduces the objectives and the methodology of this work. Section 3 presents our novel approach. Section 4 presents the architecture of the prototype to be developed. Section 5 illustrates our approach through a MDE-based VUML process, which we model using this prototype. Section 5 discusses related works, while section 6 concludes and introduces some perspectives.

2 OBJECTIVES AND METHODOLOGY

The objective of this paper is twofold: (1) provide an extension of SPEM that reifies the MDE concepts; (2) provide a prototype of a PSEE (Process-centered Software Engineering Environment) that guides process designer at modeling phase and developers at enactment time. One of the major advantages of reification of MDE concepts is to allow on one-hand process designers to explicitly describe specific aspects of MDE development, on the other hand to ensure the consistency between models produced by MDE process activities.
To fulfill the first objective, we have defined an extension of the SPEM4MDE metamodel based on the SPEM 2.0 extension mechanism.
To fulfill the second objective, we have designed a prototype of a PSEE based on a methodology that is composed of four steps (see Figure 1).

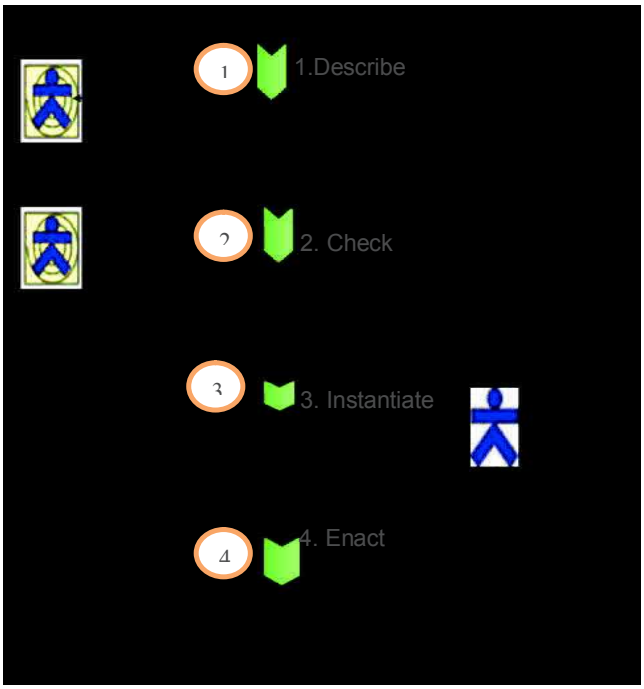

Figure 1: Methodology for modeling and enacting MDE processes

The first step, which is done by a process designer, consists in describing structural and behavioral models, which are compliant with SPEM4MDE.
Afterwards, these models are validated with respect to the OCL constraints defined in the SPEM4MDE metamodel.
The third step consists in instantiating models defined in the previous step to a specific project and assigning necessary resources for enactment (developers, rules, tools, workspaces, etc.).
The fourth and last step of this methodology is the enactment of the instantiated process model. Developers that use MDE tools realize this stage. Developers are assisted in their tasks by the enactment environment on the basis of process behavior models. This stage produces the deliverables of the project (code, models, documentation, etc.).
To validate our approach a MDE-based process for models composition VUML [1], which is dedicated to viewpoint-oriented multi-modeling has been used.

3 OUR APPROACH: THE EXTENSION OF THE SPEM4MDE METAMODEL

The extension of SPEM4MDE is structured in the form of four packages (ActivityKind, TaskUseKind WorkProductUseKind, and RoleUSeKind. They provide concepts that are able to describe any kind of process including model-driven development ones. To define their packages, we reuse the SPEM 2.0 kind-based extension mechanism. It allows process designers to refine the

SPEM concepts such as Activity, TaskUse, RoleUe and
WorkProductUse to define model-driven development processes.

3.1 ActivityKind Packaage

Figure 2 shows the *ActivityKind* package that aims at describing activities in a process (e.g. Design a system). The different kinds of activities are *Phase*, *Iteration* and *Process*. A MDE Process is a kind of process that encompasses only transformations.
The following constraint rules are used to define the static
[WF01]: A phase should be decomposed only into *Iterations*

```
Context Phase inv:
self.nestedBreakDownElement→
forAll (a: Activity | a.oclIsTypeOf
(Iteration))
```

[WF02]: An Iteration should be decomposed only into *Activities*

```
Context Iteration inv:
self.nestedBreakDownElement→ forAll (a:
Activity a.oclIsTypeOf (Activity))
```

[WF03]: An *Activity* should be decomposed only into *sub-activities*

```
Context Activity inv:
self.nestedBreakDownElement⇒
forAll (a: Activity | a.oclIsTypeOf
(Activity))
```

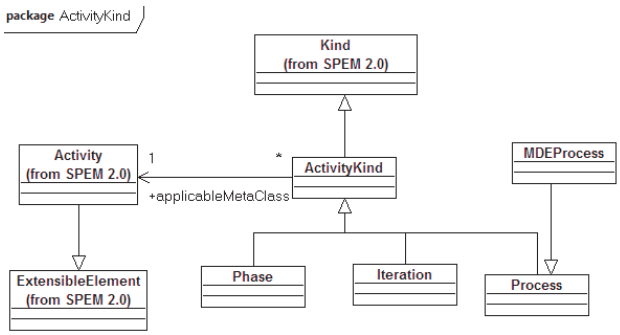

Figure 2: ActivityKind package

3.2 TaskUseKind Package

Figure 3 shows the *TaskUseKind* package that aims at describing tasks in process. A task is an activity assigned to a single person (e.g. transform a model). The different kinds of tasks are transformation and merging. It is possible to define at modeling time others kinds of TaskUse by used the extension mechanism.
The following constraint rule is used to define the static semantics of the *TaskUseKind* package.
[WF01]: input and output parameters of a transformation task should be models

```
Context Transformation inv:
self.ownedProcessParameter→collect (p|
p.parameterType)→forall(p|
p.oclIsKindOf(Model))
```

[WF02]: input and output parameters of a merging task should be models

```
Context Merging inv:
self.ownedProcessParameter→collect (p|
p.parameterType)→forall(p|
p.oclIsKindOf(Model))
```

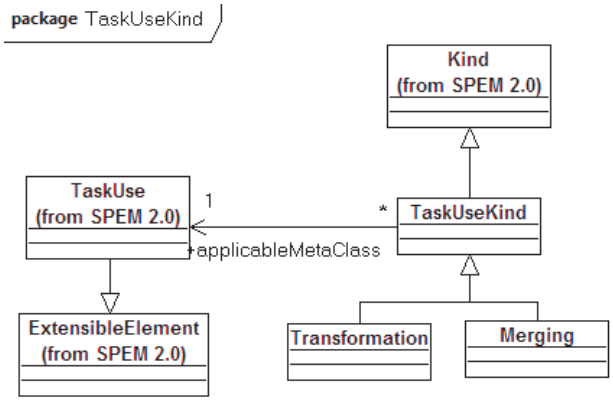

Figure 3: TaskUseKind package

3.3 WokProductUseKind Package

Figure 4 shows the *WorkProductKind* package that aims at describing any artifact/products consuming or producing within a process. The different kinds of products are models and text (code or documentation. The package describes also the relationships between models (overlap, matching, impact, compliance). For compliance relationship, a model should comply with a metamodel (i.e. a model at the *M2* level). Similarly, a metamodel should comply with a meta-metamodel (i.e. a model at the *M3* level).
The following constraint rule is used to define the static
[WF01]: WorkProducts that are used in an overlap (respectively, matching, impact) relationship should be models

```
Context Overlap/Matching/Impact inv:
self.source.oclIsTypeOf (Model))
and
self.target→ forAll (p: WorkProductUse
p.oclIsTypeOf (Model))
```

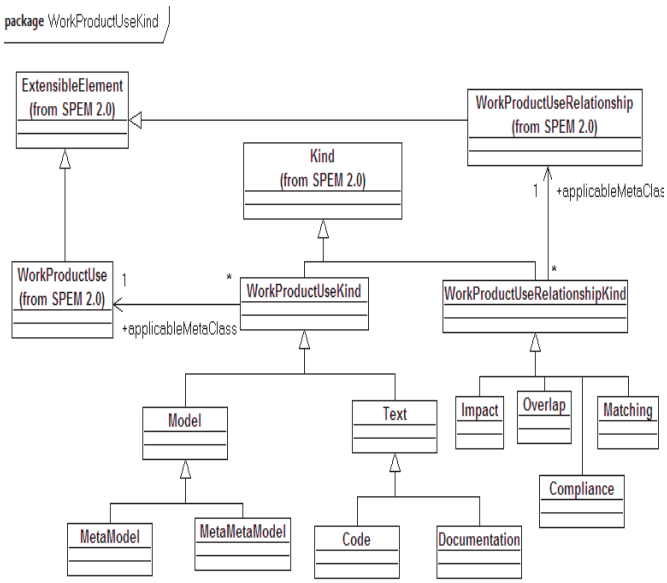

Figure 4: WorkProductUseKind Package

3.4 RoleUseKind Package

In the RoleKind package (see Figure 5), two main kinds of roles [13] are depicted: meta-level roles and application design roles. On the meta-level we distinguish the following roles: domain expert, language engineer, transformation specialist, platform expert, and software factory architect whereas the application level proposes the following roles: business engineer, solution architect, test engineer, and data modeler.

The following constraint rule is used to define the static [WF01]: A transformation specialist should participate only in transformation tasks.

```
Context TransformationSpecialist inv:
```

self.ProcessPerformer→collect(p| p.linkedActivity)→forall(p| p.oclIsTypeOf(Transformation))

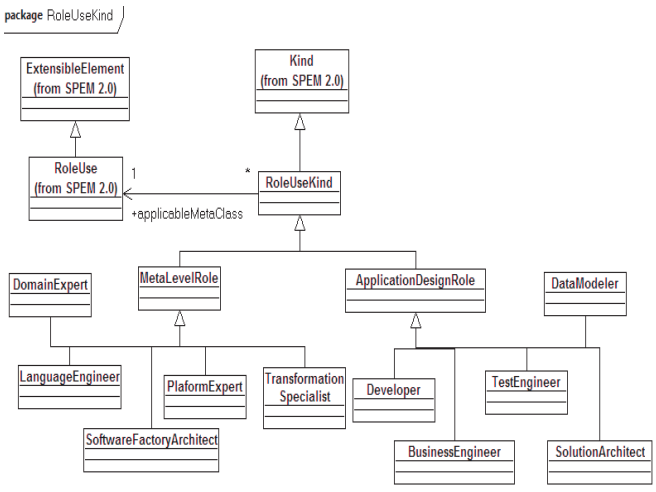

Figure 5: RoleUseKind Package

4 THE PROTOTYPE OF THE PSEE

To validate our approach, we have developed a prototype SPEM4MDE-PSEE under the PolarSys environment [31]. As shown by Figure 6, SPEM4MDE-PSEE is divided into two components: *SPEM4MDE Process Editor*, and *SPEM4MDE Process Enactment Engine.*

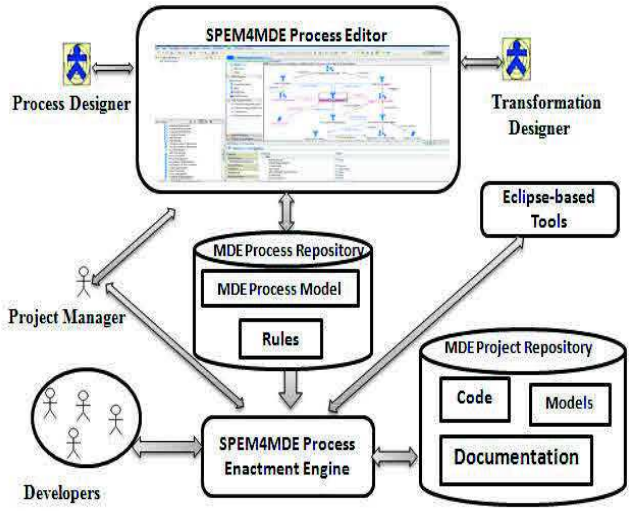

Figure 6: Architecture of SPEM4MDE-PSEE.

SPEM4MDE Process Editor allows process designers to describe, and modify process models. Describing a process model includes describing its structure and its behavior. Transformation designers describe transformation rules in a MDE process. Once the process model is described, the process designer may check it with respect to the constraints defined in the SPEM4MDE metamodel, or to additional constraints. There are two ways for checking MDE process models: checking on demand (i.e. when the user triggers himself the checking process) or checking during edition (i.e. checking is done automatically by the tool). Outcomes of process editing are stored in a repository called *MDE Process Repository*. A project manager for instantiating a MDE process model to a given project may also use this editing component.

SPEM4MDE Process Enactment Engine allows developers to enact a project-specific process model by giving them enactment operators and the current state of any process element. It is integrated with other eclipse-based tools (ATL, Smart QVT, Code Management Tool, etc.) in order to execute the activities of the instantiated MDE process. Developers can then keep track of what is the current state of each element of the MDE project, what has been done before and what is left. Outcomes (models, code, documentation, etc.) are stored in a MDE *Project Repository*

5 VALIDATION: A MDE-BASED VUML PROCESS

In this section we illustrate our approach with a MDE process example: the MDE-based VUML process for models matching and merging. First, we give an overview of the MDE-based VUML process, and after we highlight the model composition process in VUML.

VUML [25] is a viewpoint-based multi-modeling approach that

aims at reducing the complexity of design software by providing actors with a way to design a model according to their viewpoint.

The MDE-based VUML process (see Figure 7) is composed of four activities.

The first activity aims at identifying the need of each actor in order to create a requirements model (UML use case diagram).

The second activity is decentralized and consists in developing separate PIM models, each one representing a viewpoint of the desired system. The result of this phase is a set of UML models (class diagrams, state machines, sequence diagrams, etc.), which are, conform to the UML metamodel.

The third activity (see Figure **8** for detailed information) consists in composing and merging previous viewpoint models in order to generate the VUML model.

The last activity transforms the VUML model into an implementation model according to the target platform. This activity is carried out by applying an object code generation pattern as described in [24]. Finally, object code is generated from this implementation model.

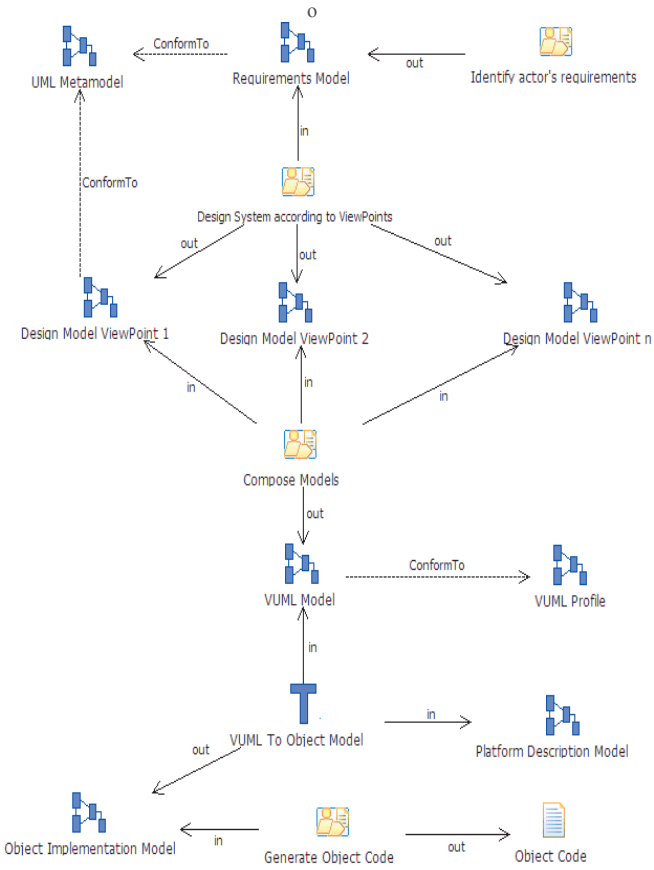

Figure 7: The MDE-based VUML Process

Figure **8** shows the different steps of the models composition activity in MDE-based VUML process.

The first step is a *precomposition* that identifies and resolves the different conflicts between viewpoint models (names, structural, etc.).

The second step (*composition*), which is automatic, aims at composing PIM models. This composition is an exogenous transformation because it takes input PIM models conform to the UML metamodel and produces an output VUML model conform to the VUML profile.

Once the VUML model is generated, the third step (*postcomposition*) consists in refining it. A reflexive relation on the VUML model represents this refinement operation. During this step, possible dependencies between the view classes of a given multiview class must be in order to ensure the consistency of the system model. These dependencies are modeled in VUML by dependency relationships, which are stereotyped by "viewDependency", and annotated by constraints expressed in OCL language.

The models composition activity in the MDE-based VUML process has been implemented with two transformation modules: the first one implements correspondence rules and

generates a correspondence model, whereas the second implements both merging and translation rules and produces the VUML target model.

The rule *Class2PartialCorrespondence* states that two class elements will be linked by a partial correspondence relationship if they are defined in two different models with the same name. This rule is declared as a specialization of *ModelElementCorrespondenceRule*, *which* is defined as an abstract rule. The inheritance mechanism allows factorizing common code among several transformation rules.

The rule *PartialCorrespondence2Base* specifies that for each defined partial correspondence relationship which links two classes, an *UML2 element* Class is created in the composed model. This rule implements the merging mechanism.

Finally, the rule *Class2Class* implements the translation rules which express that a class having no corresponding class is copied in the target model.

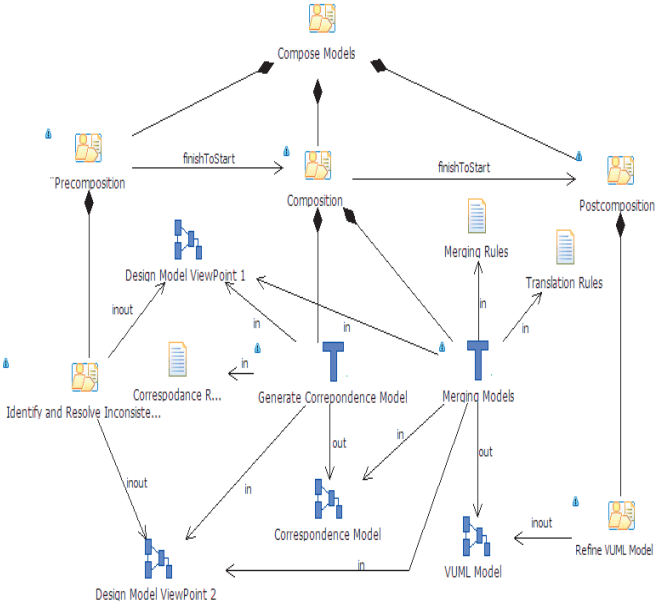

Figure 8: Models Composition in MDE-based VUML Process.

6 RELATED WORKS

Works related to our domain of interest mainly focus on transformations and MDE processes modeling. In the transformation area, we distinguish between model-to-code and model-to-model transformation approaches. In general, model-to-code can be viewed as a special case of model-to-model transformation; we only need to provide a metamodel for the target programming language. Among model-to-model transformation approaches, we distinguish between graph-based, program-based and template-based approaches. Model transformation is the central topic of MDE and is essential to define a MDE process. The first MDE process came with the OMG's MDA initiative [5, 22], which depicted a general-purpose process that can be applied to any application domain. Then, starting from the MDA approach, other MDE processes dedicated to middleware service [21], web applications [18], e-learning [33], models composition [1], embedded-systems [11], and a version of the Open Unified Process for MDD [26] have been proposed. However, there is a lack of consistent terminology since there is no unified language to specify MDE processes: each one adopts ad hoc notations and different concepts are used to define the activities and artifacts for software development life cycle.

Many languages and formalisms have been proposed for modeling software processes [3, 8, 12, 29] however only a few of them take into account natively MDE concepts [20, 27]. The SPEM 2.0 standard [29] proposes concepts that are quite generic to describe any kind of process. To overcome this limitation, SPEM proposes a kind-based extension mechanism, which we reused in this paper to define an extension of SPEM4MDE. Moreover, SPEM does not address the process enactment issue in its last version. Nevertheless, it clearly suggests two possible ways of enacting SPEM 2.0 process models: mapping the SPEM 2.0 process models into project plans or linking SPEM 2.0 process elements with external behavior formalisms. To overcome the limitations of SPEM regarding enactment, several approaches based on state-machines (eSPEM [8], xSPEM [2], SPEM4MDE [6]), Petri nets (e.g. Porres' approach [27]), and on workflow (XPDL [35], BPEL [34], BPEL4PEOPLE [34], JBPM [13]) have been proposed.

The QVT standard [23] is suitable for defining model mappings and executable model transformations, but fails on describing process design aspects.

In [6] we present a first version of the SPEM4MDE metamodel. It extends a subset of SPEM 2.0 by adding natively concepts and semantics relating to MDE. In addition, SPEM4MDE offers a set of behavioral models described with UML state-machines that process designers may reuse or adapt for a particular process. However, SPEM4MDE does not provide a mechanism that allows to refine its own concepts for a desired MDE process.

In [27], an approach that is targeted towards the development of software and systems using MDE methods is presented. The dynamics of this approach is based on Petri Nets. This approach can be integrated with existing approaches for software process modeling, but the metamodel contains only one concept (*transformation Tool*) that is related to MDE.

In [20] an approach to MDA process specification, based on the SPEM 2 standard concepts, is proposed. A tool called Transforms, which can be used to instantiate a MDA process for a given domain, supports this approach. Developers can describe steps and associate artifacts to perform MDA modeling and transformations chain. This approach has however some limitations, since it is tightly coupled with MDA concepts. Furthermore, it does not separate the specification of a transformation from its implementation.

7 CONCLUSION

In the literature, a few PML (Process Modeling Language) are
natively supported MDE concepts. Reification allows promoting MDE concepts as first-class citizens.

In this paper, we have presented an extension of SPEM4MDE by using the SPEM 2.0 kind-based extension mechanism. This mechanism allows process designers to refine SPEM concepts in order to flexibly define any kind of process particularly MDE ones.

To validate our approach, we developed a prototype and used the MDE-based VUML process for models composition.

Two important perspectives of this work are under consideration. Firstly, we intend to use the MDE-based VUML process for the development of an emergency management application within a senegalese hospital. Secondly, we envisage extending the SPEM4MDE metamodel for handling the execution of model-based collaborative development processes.

ACKNOWLEDGMENTS

This work was partially supported by CEA-MITIC.

REFERENCES

[1] Anwar, A., Ebersold, S., Nassar, M., Coulette, B., Kriouile A., 2010. A Rule-Driven Approach for composition of Viewpoint-oriented Models. In: JOT (Journal of Object Technology/

[2] Bendraou, R., Combemale, B., Crégut, X.and Gervais, M.-P., 2007: "Definition of an eXecutable SPEM2.0". In: 14th Asia-Pacific Software Engineering Conference (APSEC), pp. 390-397. IEEE Computer Society, Nagoya, Japan.

[3] David A.Anisi. 2003. *Optimal Motion Control of a* Bendraou, R., Gervais, M.P., and Blanc, X., 2005. "UML4SPM: a UML 2.0-based metamodel for software process modeling". In: Briand, L., Williams, C. (eds.) MODELS 2005. LNCS, vol. 3713, pp. 17-38. Springer, Montego Bay, Jamaica

[4] Bézivin, J., and Breton, E., 2004. "Applying the basic principles of model-engineering to the field of process engineering". European Journal for the Informatics Professional. 5, 27-33.

[5] Bézivin, J., and Gerbé, O, 2001. "Towards a precise definition of the OMG/MDA Framework". In: Proceedings of the 16th IEEE international conference on Automated Software Engineering (ASE), pp. 273. IEEE Press, San Diego, USA.

[6] Diaw, S., Lbath, R., and Coulette, B., 2011. "Specification and Implementation of SPEM4MDE, a metamodel for MDE software processes" (regular paper). In: International Conference on Software Engineering and Knowledge Engineering (SEKE 2011), pp. 646-653. Knowledge Systems Institute, Miami, USA.

[7] Fall Ibrahima; Diaw Samba: A Metamodel for MDE Process Model-Products Relationships. 2016 IEEE 25th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE) Pages: 166 – 171.

[8] Elner R., Al-Hilank, S., Bediaga, A., Drexler, J., Jung, M., Kips, D., and Philippssen, M., 2010. "eSPEM – A spem extension for enactable behavior modeling". In: Kuhne, T., Seloc, B., Gervais, M.P., Terrier, F. (eds.) ECMFA 2010. LNCS, vol. 6138, pp. 116-131. Springer, Paris.

[9] Fondement, F., and Silaghi, R., 2004."Defining model-driven engineering processes". In: 3rd UML Workshop in Software Model Engineering (WiSME), Springer, Lisbonne.

[10] France, R., and Rumpe, R., 2007. "Model-drive development of complex software: A Research Roadmap" In: Proc. of the International Conference on Software Engineering (ICSE), pp. 37-54. IEEE Press, Minneapolis, Minnesota, USA.

[11] Garcia, A., Combemale, B., Crégut, X., Guyot, J.N., and Libert, B., 2008. "TopProcess: A process model-driven approach applied in Topcased for embedded real-time software". In: European Congress on Embedded Real-Time Software (ERTS), Société des Ingénieurs de l'Automobile, Toulouse

[12] Gonzalez-Perez. C. "Supporting Situational Method Engineering with ISO/IEC 24744 and the Work Product Pool Approach". Situational Method Engineering: Fundamentals and Experiences, pp. 7–18, 2007.

[13] JBPM-website: www.jbpm.org

[14] Hann , J. D., 2009. Roles in Model-Driven Engineering. Available at: http://www.theenterprisearchitect.eu/archive/2009/02/04/roles-in-model-driven-engineering

[15] Humphrey, W., and Kelner, M., 1989. "Software modeling: principles of entity process models". SEI - Carnegie Mellon University. Pittsburgh, Pennsylvania

[16] Kent, S., 2002. "Model-driven engineering". In: Grieskamp, W., Santen, T., Stoddart, B. (eds.) IFM 2002. LNCS, vol. 2335, pp. 286-298. Springer, Turku, Finland

[17] Kleppe, K., Warmer. J., and Bast, W., 2003. "MDA explained the model-driven architecture: practice and promise", Addison-Wesley

[18] Koch, N., 2006. "Transformations techniques in the model-driven development process of UWE". In: 6th International Conference on Web Engineering (ICWE), Volume 155 Article N° 3. ACM, California

[19] Larrucea, X., García Díez, A. B., and Mansell, J. X., 2004. "Practical model-driven development process". In: Second European Workshop on Model Driven Architecture (MDA) with an emphasis on Methodologies and Transformations, pp. 99-108. Computing Laboratory, University of Kent, Canterbury, UK

[20] Maciel, R.S.P, Silva B.C., Magalhães, A.P.F., and Rosa, N.S, 2009. "An approach to model-driven development process specification". In: 11th International Conference on Enterprise Information Systems ICEIS), pp. 27-32. INSTICC Press, Milan

[21] Maciel, R.S.P, Silva B.C., and Mascarenhas, L. A., 2006. "An edoc-based approach for specific middleware services development". In: 4th Workshop on MBD of Computer Based System, pp.135-143. IEEE Press, Postdam.

[22] MDA_Spec at, http://www.omg.org/mda/executive_overview.htm

[23] MOF 2.0 QVT 1.0_Spec, http://www.omg.org/spec/QVT/1.0

[24] Nassar M., Anwar, A., Ebersold, S., El Asri, B., Coulette, B., Kriouile, A. Code Generation in VUML profile: a Model Driven Approach. IEEE/ACS AICCSA 2009. Rabat, May 10-13, 2009. IEEE Computer Society Press.

[25] Nassar, M., Coulette, B., Crégut, X., Ebersold, S and Kriouile., A. Towards a View based Unified Modeling Language. ICEIS'03, Angers, France, 2003.

[26] OPEN UP at: www.eclipse.org/epf/openup_component/mdd.php

[27] Porres, O., and M. C. Valiente, 2006. "Process definition and project tracking in model-driven engineering". In: Münch, J., Vierimaa, M. (eds.) PROFES 2006. LNCS, vol. 4034, pp. 127-141. Springer, Amsterdam.

[28] Rios, E., Bozheva, T., Bediaga, A., and Guilloreau, N., 2006. "MDD maturity model: a roadmap for introducing model-driven development". In Rensink, A., Warmer, J. (eds.) ECMDA-FA 2006. LNCS, vol. 4066, pp. 78-89. Springer, Bilbao (2006)

[29] SPEM 2.0_Spec, http://www.omg.org/spec/SPEM/2.0

[30] Stahl, T., and Volter, M., 2006. "The Model-driven software development Technology, Engineering, Management", Translation copyright by John Wiley & Sons, Ltd

[31] PolarSys_website, www.polarsys.org

[32] UML 2.2_Spec, http://www.omg.org/spec/UML/2.2/

[33] Wang, H., and Zhang, D., 2003. "MDA-based development of E-Learning system". In: 27th International Computer Software and Applications Conference (COMPSAC), pp. 684-689. IEEE Press, Texas.

[34] Web Services Business Process Execution Language Version 2.0. Working WS-BPEL TC OASIS, April 2007. URL: http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf.

[35] XPDL-Spec, www.omg.org/bpmn/.../XPDL_BPMN.pdf.