Autom Softw Eng (2013) 20:427–461
DOI 10.1007/s10515-013-0124-0

Supporting model-driven development using a process-centered software engineering environment

Rita Suzana Pitangueira Maciel · Ramon Araújo Gomes · Ana Patrícia Magalhães · Bruno C. Silva · João Pedro B. Queiroz

Received: 1 April 2012 / Accepted: 19 March 2013 / Published online: 10 April 2013

Abstract The adoption of Model-Driven Development (MDD) is increasing and it is widely recognized as an important approach for building software systems. In addition to traditional development process models, an MDD process requires the selection of metamodels and mapping rules for the generation of the transformation chain which produces models and application code. In this context, software process tasks should be performed in a specific sequence, with the correct input artifacts to produce the output ones. However, existing support tools and transformation engines for MDD do not have a process-centered focus that addresses different kinds of software process activities, such as application modeling and testing to guide the developers. Furthermore, they do not enable process modeling nor the (semi) automated execution of activities during process enactment. The MoDErNE (Model Driven Process-Centered Software Engineering Environment) uses process-centered software engineering environment concepts to improve MDD process specification and enactment by using a metamodeling foundation. In MoDErNE, a software process model may be enacted several times in different software projects. This paper details the MoDErNE environment, its approach and architecture and also the case studies through which the tool was evaluated.

R.S.P. Maciel (✉) · R.A. Gomes · A.P. Magalhães · B.C. Silva · J.P.B. Queiroz
Computer Science Department, Federal University of Bahia, Salvador, Bahia, Brazil
e-mail: ritasuzana@dcc.ufba.br

R.A. Gomes
e-mail: ramon@dcc.ufba.br

A.P. Magalhães
e-mail: anapfmm@dcc.ufba.br

B.C. Silva
e-mail: brunocs@dcc.ufba.br

J.P.B. Queiroz
e-mail: jpqueiroz@dcc.ufba.br

Keywords Software process · Model-driven development · Process centered software engineering

1 Introduction

Over the years, several software development supporting environments have been proposed and used in practice as tools for the development process. Some of these environments are dedicated to supporting specific tasks for a software development process. For example, while an IDE (Integrated Development Environment) supports software codification in a given programming language, a diagram editor supports software modeling. In a single software development project several tools are generally needed to support software engineering activities. Throughout a software project, developers should know their tasks, when to perform them and which tools to use. The project manager should have an overview of the software process tasks, as well as be aware of all the workflow of the tasks, the roles involved, income and outcome artifacts, supporting tools, and any other important information about the process elements and their relationships.

Therefore, all the process elements and their relationships can be described by a given notation able to define what we call a Process Model. Process models should define which software development activities are expected to be executed, when to perform them and by whom. Process models should also identify tools to be used and the format of documents to be created and manipulated. Software process modeling should facilitate the communication, understanding, reutilization, evolution, management and standardization of the process (Humprey and Kelner 1989). A process model is said to be enacted when a development team follows the process model definitions during the development life cycle.

The modeling of software processes is a non-trivial and time-consuming task which means that defining a new process for each software project is unfeasible. One of the key issues for supporting software processes is how software process models and software engineering environments are related and how a supporting infrastructure can be derived from the information given in a software process model (Gregor et al. 2001). This environment should support various types of activities involved in software development and the literature usually refers to such an environment as a PSEE (Process-centered Software Engineering Environment) (Gregor et al. 2001; Ambriola et al. 1997). By using PSEEs a process has to be modeled first and then it can be enacted in several projects. During process enactment, software developers are reminded of activities which have to be carried out, automatic activities may be executed without human interaction and consistencies among documents are facilitated. In this context several PSEEs have been proposed (Arbaoui et al. 2002; Lima et al. 2006; Montoni et al. 2006; Mohagheghi and Dehlen 2008), but many of these have limitations such as restricted support for a specific software development process, a focus restricted to management tasks while lacking features for enactment support. Several PSEE concepts proposed in the literature have not been incorporated as such in the first generation of PSEE environments but they have influenced the understanding of software processes and infrastructure needed to support it substantially

(Gregor et al. 2001). Consequently, several studies suggest a set of requirements and evaluation frameworks for these environments in order to guide the development of environments that wish to support software processes properly (Atkinson et al. 2001).

Apart from PSEE proposals, software development techniques are continuously evolving in order to solve the main problems that affect the development and maintenance of software systems (Koch 2006). Model Driven Development (MDD) is an approach that is primarily concerned with reducing the gap between the problem and solution spaces. More specifically, the application of MDD relies on software implementation domains through the use of technologies that support systematic transformation of problem-level abstraction in software implementations (France and Rumpe 2007). System models are not only used for system documentation, but they actually serve as a basis for the implementation phase. Each activity in the development process requires a number of input models that produce further models as output. In this way, the development of an application can be viewed as a set of transformations that lead to the final system. MDD has changed not only the way systems are built but also the way they are tested (Mussa et al. 2009). Model Driven Testing (MDT) (Baker et al. 2007) is an approach based on MDD in which tests can be generated from development models in an automated way through the use of transformations. One of the most well known initiatives in this scenario is Model-Driven Architecture (MDA) proposed by Object Management Group (OMG) (OMG 2003). MDA relies on several OMG standards to perform MDD concepts. As MDA is an MDD realization, in this text we use only the MDD acronym for software processes that use this approach, including the ones which follow OMG standards.

Unlike traditional development process models (Rational Unified Process (RUP), eXtreme Programming (XP), Open/UP, etc.), an MDD process requires the selection of metamodels and mapping rules for the generation of the transformation chain which produces models and application code. In this context, if modeling and transformation tasks are not properly performed, the desired final code will not be reached. Thus, existing research in MDD practice has revealed the importance of software processes and suitable tools, concluding that they are crucial for the use of the MDD approach in industry (Hutchinson et al. 2011a). The necessary techniques to apply MDE correctly depend on tool support and integration in the software project (Hutchinson et al. 2011a, 2011b).

Many tools have been designed to support MDD. These environments usually have a specific focus on a transformation strategy or transformation engine in order to automatically generate models, codes and test cases from a variety of models. However, current MDD supporting tools are basically interested in defining and executing transformations which produce code and deployment artifacts from models (e.g. AndroMDA,[1] Blu Age[2] and others) for a specific part of the software life cycle or for a specific domain. Indeed, other activities in a software process are usually not considered. They do not focus on the software process specification, neglecting support for the integration of different process specification activities into the software development process phases. On the other hand, tools for process modeling and

[1] AndroMDA.org Home—http://www.andromda.org/.

[2] Blu Age—http://www.bluage.com/en/en_home.html.

specification (e.g. EPF[3]) lack integration to modeling tools and model transformation engines. This scenario does not help software engineers who would like to use MDD as a main software development approach or to adapt existing software processes.

This paper presents MoDErNE (Model Driven Process-Centered Software Engineering Environment) which is an environment for model driven development that uses process centered concepts to aid the adoption of MDD. The environment is based on the MDD and MDT approaches for both system development and testing, using SPEM (Software and Systems Process Engineering Metamodel) (OMG 2008) concepts. Based on metamodels, model-driven processes can be instantiated. Therefore this instance, a specific process model, can be enacted several times in different software development projects.

By using and combining process-centered software engineering concepts and a model-driven approach, our project offers an environment that supports model-driven software process specification and enactment properly. It offers an environment where different strategies for modeling and transformations tasks can be specified and used in a proper sequence and in an integrated way as we aim to assist and facilitate the use of the MDD approach in software processes.

MoDErNE approach was initially proposed in Maciel et al. (2009). The first version of this tool, named *Transforms* (Silva et al. 2009), was only able to support software development tasks. Currently, it comprises two modules with complementary features: a Process Editor and a Process Executor. In the Process Editor, it is possible to specify process models using a specialization of the SPEM metamodel. It is possible to specify several kinds of software development, testing and management activities. In the Process Executor, it is possible to enact the processes previously specified in the Process Editor using UML (Unified Modeling Language) diagrams and transformations.

This paper is organized as follows: Sect. 2 presents related work; Sect. 3 describes MoDErNE, the proposed solution; Sect. 4 focuses on case studies; four case studies are used to evaluate MoDErNE. In this section, we present a summary of the results of the first two case studies, while the latter two are presented in more detail. A more detailed description of the first two can be seen in Maciel et al. (2009). Section 5 presents final remarks and future work.

2 Related work

In recent years a number of research initiatives related to MDD have emerged. We can divide these into two categories which we explain in this section: processes and methodologies for MDD; and languages and tools for model transformation. We also describe some related work about PSEEs and how their legacy concepts connect to our tool environment.

[3]Eclipse Process Framework—http://www.eclipse.org/epf/.

2.1 Processes and methodologies for MDD

Several methodologies have been proposed in the literature. Some of these include MDA process for middleware specific services (Maciel et al. 2006), MDD process for web applications (Koch 2006), MDA methodology for e-learning systems (Wang and Zhang 2003), fault tolerance distributed software families (Guelfi et al. 2003), KobrA (Atkinson et al. 2001) a method for the development of component-based software with model driven techniques. The MDD approach to software development has not only changed the way software systems are built and maintained but also the way they are tested. In this context, several works have been proposed concerning Model Driven Testing (MDT). Most of them (Hartmann et al. 2004; Yuan et al. 2008; Mingsong et al. 2006; Bouquet et al. 2008) propose methods to generate test cases from system models in order to increase automation in testing activities. In Javed et al. (2007), for instance, the authors aim to generate test models from system models, using use case and activity diagrams, annotations and transformation rules to derive the test cases.

However, most of the approaches for MDD and MDT processes and methodologies are defined using non-standard notation and language. Most of them are specified imprecisely in natural language with supplementary pictures and diagrams. In fact, there is a lack of consistent terminology as there is no unified language to specify MDD processes. Each one adopts *ad hoc* notations and different concepts are used to define the activities and artifacts for the software development life cycle. In addition, each methodology is focuses on a specific application domain and several of them comprise only a specific software project phase (e.g. architectural design, testing, etc.). In this scenario the use of different strategies for a given software process makes it difficult. Software process modeling using unified and consistent terminology should make understanding, reutilization, management and standardization of the process possible (Humprey and Kelner 1989).

2.2 Languages and tools for model transformation

The second research initiative is related to model transformation. Tool support plays an important role in software development activities and in the model-driven development context this is no different. It is essential and even more necessary due to the intrinsic need for automatic model transformation and code generation. Therefore organizations which follow a model-driven approach also have to use a supporting tool to automate, even partially, their transformations. The process for model-driven development must then be accomplished with minimum tool support; otherwise it could become unfeasible (Hutchinson et al. 2011a).

Transformation is an essential activity for model-driven development, many of the approaches in the literature focus on this task. Several languages for model transformation specification have been proposed as well as a number of transformation engines to carry out the transformations. At present, there is a variety of open source and proprietary MDD languages with different characteristics and features, such as:

ATL (Atlas Transformation Language)[4] and QVT (Query-View-Transformation)[5] for model-to-model transformation; and MOFScript[6] and Acceleo[7] for model-to-text transformation.

Other environments enable system modeling and provide predetermined transformations for specific platforms, programming languages and also for a specific phase for software development life cycle (requirement, architecture, codification, etc.). For example AndroMDA is an environment which follows the MDA approach and provides transformations from platform independent models (PIM) to the Java EE platform. The transformations are provided in the so called cartridges and are added as plug-ins for the environment. Another example is BluAge, which is a proprietary tool that claims to facilitate the migration of legacy systems. The tool can extract platform independent models from a legacy system code and provides transformation to generate code for a different platform, such as Java EE or.NET. WebRatio is another proprietary tool used to develop web applications. It uses BPMN (Business Process Management Notation) and WebML (Web Modeling Language) to represent the business logic and system requirements and then generates java code for the Java EE platform. Other MDD tools can be easily found on the web, such as in http://www.modelbased.net/.

In spite of the high number of MDD tools already proposed as well as those used both in academia and industry, most focus primarily on model transformation execution, i.e. they are interested in defining and executing transformations which produce code and deployment artifacts from models. Therefore they are used in specific tasks and do not cover the whole development process life cycle. A development process involves other important tasks which should be carried out during the process enactment such as requirement analysis, testing, manual tasks etc. rather than just doing model transformations. Recently, SPEM4MDE (Samba et al. 2011) has been proposed. Similar to MoDErNE, SPEM4MDE approach uses SPEM as PML and its tool also has a Process Editor and Enactment modules. For instance, the approach focus relies on transformation definition and execution. The environment does not support MDA and MDT specificities and its does not have several of PSEE desired features, such as management tasks. Although transformations are an important aspect in MDE processes, it is a part of these processes.

Recent studies about MDE practice reveal some issues that should be addressed (Hutchinson et al. 2011a, 2011b; Mohagheghi and Dehlen 2008). Successful MDE adoption appears to require a progressive and iterative approach as it is usually adopted gradually, in specific stages or tasks of a software development process (testing, coding, etc.). Despite the fact that software processes are recognized as being important in successfully applying MDE, proposed model-driven methodologies or processes have been considered unsuitable for use by participants of these studies. Additionally, much effort is required to develop new transformations or customize existing ones (Hutchinson et al. 2011b). Some findings point to specific issues with

[4] ATL—http://www.eclipse.org/atl/.

[5] QVT—http://www.omg.org/spec/QVT/1.0/.

[6] MOFScript—http://eclipse.org/gmt/mofscript/.

[7] Acceleo—http://www.acceleo.org/.

regard to MDE tools, for example, they are expensive or need to be used in specific ways (Hutchinson et al. 2011b). The decision to adopt an MDE approach is not made with much understanding of the necessary process change (Hutchinson et al. 2011a).

PSEE is a process centered environment which gives support to various types of activities during the software development life cycle. It provides many services for the software developers by modeling and enacting already modeled processes. Interest in the PSEE approach is not new. While the first generation of PSEE environments and characteristics was revised in Arbaoui et al. (2002) and Gruhn (2002), the latest generation (after 2003), proposed in the last ten years has been analyzed in Reza (2012). Initially, some PSEE desired requirements include process modeling through a Process Modeling Language (PML), process models enactment, process tasks ordering, process evolution, management and documentation. Several environments have been proposed (e.g. Cass et al. 2000; Weber et al. 2009; Zamli et al. 2005) however, they have limitations in supporting process enactment and integrating different kinds of tools. To solve this problem, Gruhn (2002) proposed a middleware solution focus.

Although a middleware platform approach was proposed as a solution, modern PSEE prefers to focus on a certain software process domain (for web application, model-driven, software product lines) to follow the strategies of software process definition and then its enactment. The common functionalities of the latest PSEEs (e.g. SPACE (Weber et al. 2009), VRML (Zamli et al. 2005), WebApSEE (Lima et al. 2006), Transforms (Silva et al. 2009), etc.) (Reza 2012) are: interactive assistance throughout software development, automation of routine and labor-intensive tasks and invocation and control of software development tools, process flexibility during the enactment and software team distribution. However, they have failed to provide security and mobility features. Besides these general functionalities, an MDD focused PSEE should allow users to represent MDD related process elements, such as transformation and model artifacts. Furthermore, it should provide an enactment environment that supports both system modeling and model transformation, which are essential activities within the MDD context. To the best of our knowledge, there is no such environment available either in academia or in industry.

Currently MDD tools do not have process-centered features as they are not designed for this, and current PSEEs do not address specific features for MDD processes. We believe that it is important to have a tool which can help to define the activities, artifacts and roles of the software development process within the MDD context as well as integrate different approaches and tools that automate model and transformation tasks and thus support the process enactment properly.

Our work attempts to provide an environment that supports both process modeling and enactment for the MDD process context. In the model driven approach, system models are metamodel instances. Our approach uses this same strategy for software process specification. It comprises a set of metamodels to provide a standard notation to specify a software process allowing the explicit representation of MDD related elements, such as models and model transformations. Process element descriptions are placed at the metamodel level, and consequently process models become instances of these metamodels. The approach has several other PSEE characteristics adapted for the MDD context, such as (semi) automation support for process activities, a collaborative environment and integration with other MDD related tools.

3 MoDErNE

MoDErNE is an approach and tool which supports the specification and enactment of model-driven software processes in an integrated way using process-centered software engineering concepts. Figure 1 gives an overview of the MoDErNE environment. It has two main goals: process specification, through Process Editor Module; and process enactment, through Process Executor Module.

Process Editor comprises a UML profile editor; a transformation rule editor; and process publication. The Process Executor comprises process enactment in collaborative way following the previous process specification; UML modeling editor; and transformation execution for code generation.

The MDD process specification includes definitions of management, development and testing processes. These definitions follow a set of metamodels based on SPEM 2.0. The result of this is that MDD process models, which can be expressed as an SPEM instance and which uses UML as a modeling language, may use this environment as a support tool. Additionally, we extended SPEM to adapt them to the MDA context, e.g. Computation Independent Model (CIM), Platform Independent Model (PIM) and Platform Specific Model (PSM) concepts.

The process is represented by models, which are instances of these metamodels and they are basically made in UML diagrams. The process may be modeled by: (i) a class diagram, to show the process elements in a visual representation; (ii) an activity diagram to model phase/iteration sequence and their specific tasks (iii) a use

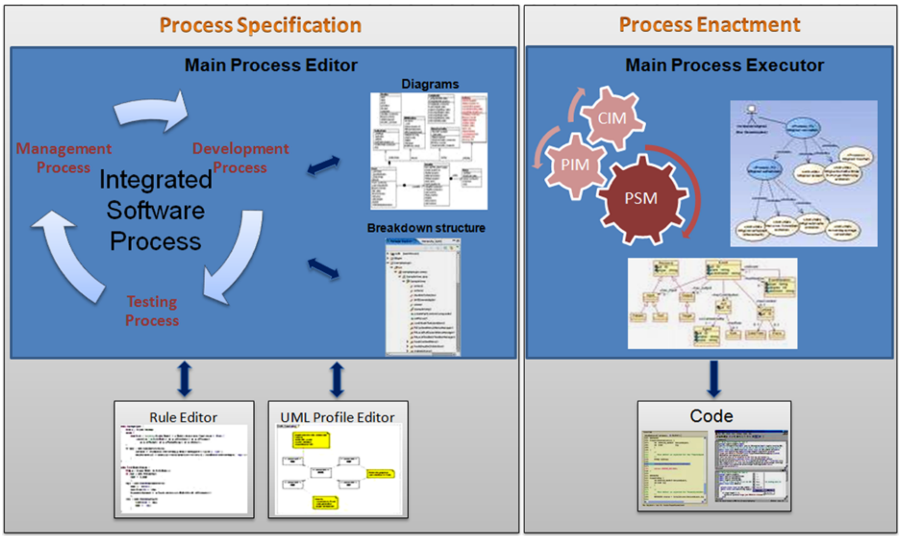


Fig. 1 MoDErNE's main functionalities

case diagram to map responsibilities through associations between roles and tasks. As well as this, a work breakdown structure is also available for those who prefer a hierarchical visualization.

The Process Editor enables the creation of reusable software engineering best practice libraries. For example, some of methodologies proposed and cited in Sect. 2 could be specified as a library. Thus, libraries can be used for: web applications, real-time systems, testing services, etc. These libraries comprise elements such as tasks, steps, roles and produced/consumed artifacts (i.e. UML models and transformation rules) for the process specification. The definition of a new process begins by selecting library elements and organizing them in terms of phases and iterations, besides the common process elements (i.e. phases, tasks and roles). MDD processes need some new definitions such as model transformation and profiles. These definitions are supported as the environment provides a transformation rule editor (ATL, MOFScript and QVT for instance) and a UML profile editor respectively. Process definitions are stored in a repository to be later enacted by the Process Executor module.

To exemplify the use of the Process Editor, it is useful to consider an organization that wants to adapt their RUP process for the MDD approach. In the process editor we should specify the reusable elements such as disciplines (e.g. Requirements and Analyze & Design), tasks (e.g. define requirements, define scope and so on), the workproducts involved (e.g. use case model, class model) and the performed roles (e.g. system analyst). We should also specify dynamic definitions such the phases (inception, elaboration, construction and transition phases) and iterations (e.g. the elaboration phase has two iterations) using the tasks previewed modeled. Besides these definitions, MDD processes require other specifications: it is also necessary to develop the appropriate transformations for the process and select the source and target profiles (e.g. we can develop a transformation to transform inception phase model the first version of elaboration the phase model). These activities also involve new roles (e.g. process specifier, transformation specifier and transformation developer) that only exist in MDD processes. The resulting process, named RUPMDD, was then organized into two different libraries: (i) *ruplib* and (ii) *inception2elaboration*. *Ruplib* keeps the RUP process definition and *inception2elaboration* keeps the ATL transformations files and UML profiles that enable the model transformations from one phase to another. The specification of the sequence in which process tasks from these two libraries must be performed are also defined using MoDErNe Process Editor support.

The main goal of the Process Executor is the enactment of a process previously specified in the Editor and stored in the repository. Phases, iterations and their tasks are followed in the sequence as specified in the process editor. The execution of these tasks is managed by the Executor showing a task status that indicates if the task is finished, in progress, pending, etc. The Process Executor integrates modeling and runs transformation tools. When executing a modeling task, a modeling editor (i.e. UML2Tools) is available with the appropriate diagram elements and profiles specified for the task. When executing a transformation task, a transformation tool is presented (i.e. MOFScript or ATL engine) to run the rules specified in the process. Following the process tasks sequence the developer can create his/her models and transform them until code generation. Any other modeling tool can be used and import the XMI (XML Metadata Interchange) file to use as input in a transformation rule. As the process comprises validation tasks, test cases may also be generated for software validation. MoDErNE also allows the development team to work in a collaborative way to build a software system. When starting software development a team is defined according to the roles specified for the process and associated to each task. Each person in this team can execute different tasks at the same time during process enactment.

Any kind of task may be specified for the process and will be available in the sequence when enacted. However, only modeling tasks (producing UML workproducts) or transformation tasks (related to transformation workproducts) are associated to the UML editor and transformation engines. Using our example of the RUPMDD process, the Process Executor shows the inception, elaboration, construction and transition phases sequentially with their iterations and tasks. When a user (associated to a specific role) selects a task to perform, the appropriate tool is shown according to the Process Editor specification. For example, when *system analyst* performs the *define requirements* task a UML editor is automatically opened with the appropriate metamodel for the use case diagram specification, when a transformation task is selected, the associated transformation is executed, and so on.

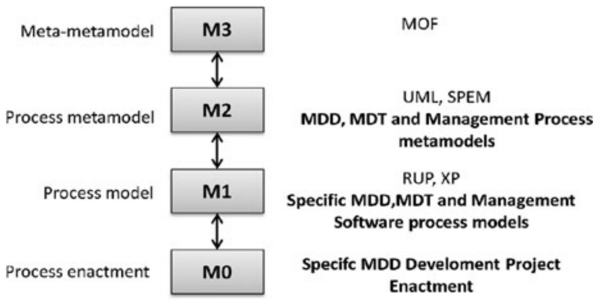
The following sections explain our approach in greater detail. Section 3.1 focuses on the metamodeling foundations on which MoDErNE is based. Section 3.2 explains the tool architecture and Sect. 3.3 details its main modules and functionalities.

3.1 MoDErNE's metamodels

In this section we present the metamodels on which MoDErNE is based and that represent software processes that use MDD techniques. They make the modeling and instantiation of MDD, MDT and management processes elements explicit and have specialized semantics that facilitate automated support during the process enactment. Just as metamodels are used to describe application models of the same domain, they can also describe the software processes models that guide the development of these applications. Therefore, our main goal is to provide a mechanism with a metamodeling foundation in order to create an effective way to support a model-driven development process specification and enactment.

According to the OMG model layers shown in Fig. 2, a specific software development project is located at level M0, i.e. the layer where a development team works on a project enacting a process which is specified in the level above (M1). RUP, XP

Fig. 2 OMG Model layers


Meta-metamodel
M3
MOF
Process metamodel
M2
UML, SPEM
MDD, MDT and Management Process metamodels
Process model
M1
RUP, XP
Specific MDD,MDT and Management Software process models
Process enactment
M0
Specifc MDD Develoment Project Enactment

and other processes are situated at M1. Process models at M1 are designed according to a process metamodel (i.e. a metalanguage to specify process models) which corresponds to level M2. For instance, SPEM was used to design the well-known RUP process model. As highlighted in Fig. 2, our approach is located at level M2. Thus, an MDD process model (located at level M1) can be designed and will be available for the development of new projects at level M0. Consequently, any process definition modeled in M1 can be used during the process enactment in M0 providing specific features according to it specification. The definition of MDD, MDT and management process concepts at metamodel level (M2) is important to provide a meaningful way to design software processes with explicit characteristics of these kinds of processes. The specified semantic for each element is important because it will be used in the process enactment (i.e. use the profile associated to a specific phase to validate the input models, associate a transformation engine to a transformation rule artifact, etc.).

Our approach is composed of a set of metamodels, based on SPEM 2.0, which refer to each aspect of a software process (development, testing or management) in an MDD context. According to SPEM, software process specification should be divided into two dimensions: static concepts, which are made up of disciplines, tasks, steps, roles and *workproducts*, forming what they call *Method Content*; and dynamic concepts, which include phases, iterations and *taskuses*, forming the so-called *Process*.

The *Method Content* involves co-related elements that can be reused in many process models. We specialize a *MethodContent* in three different types, representing the various processes needed in a software development lifecycle: the *Core Method-Content*, which represents general MDD development process elements; the *Method-ContentForTesting*, which describes testing process elements following the MDT approach; and the *ManagementMethodContent*, which focuses on project management based on PMBOK.

It is therefore possible to specify the development, testing and management processes independently. This can increase the potential reuse of one kind of process with others. For instance, the same testing process (specified in the *MethodContent-ForTesting*) could be used with many different development processes (located in the *Core Method Content*). This independent process specification can also facilitate the work of independent teams for development, testing and project management.

After modeling each process independently, they have to be integrated in a unified process that covers all software development aspects (Fig. 3).

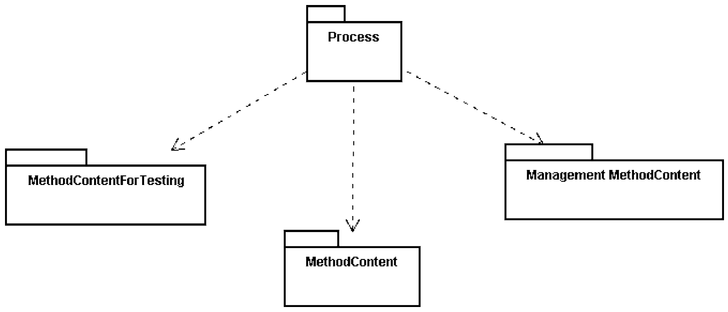


Fig. 3 Software process integration

As explained before, the *Process* is used to represent dynamic aspects of software processes, such as phases and iterations. It references the static elements defined in a *MethodContent* package to perform a complete process specification. A software process instantiated in a *Process* package can use elements of different kinds of *MethodContents* and can therefore reference development, testing and management elements. Thus, the *Process* package is responsible for the process integration, selecting elements of each *Method Content* and distributing them to the phases and iterations of a process instance.

A model-driven development process called OpenUP/MDD (OpenUP Component 2008) is a variation of the Open Unified Process for MDD. This process was specified according to the SPEM 2.0 standard using the EPF tool,[8] which is an environment for software process modeling following SPEM. As a result, OpenUP/MDD is an instance (i.e. a metamodel instance) of the SPEM metamodel. Unlike this process, we decided to add the MDD concepts at the metamodel level.

Our hypothesis is that by using metamodeling techniques to describe several aspects of software processes, they can be integrated in a flexible manner enabling better software process specification and enactment.

The following subsections explain each metamodel that forms part of our approach.

3.1.1 Core metamodel

The use of MDD requires process definitions associated with modeling activities and transformation rules to compose the transformation chain. These elements are not usually found (explicitly) in traditional software development processes. Therefore, we selected some of the SPEM 2.0 concepts and specialized them in order to cover specific aspects of the MDD context. Again, it is important to highlight that our hypothesis focuses on the explicit modeling and instantiation of process elements with specialized semantics which can facilitate the process design and enactment. The core metamodel is illustrated in Fig. 4.

As explained before, the *Method Content* package represents the process static elements. A *Discipline* groups a set of related *Tasks* that are performed by *Roles*. A *Role* defines the responsibilities of an individual or a group of individuals. A *Task* may comprise many *Steps* to describe meaningful work. During the process enactment, *Workproducts* as input and output artifacts can be consumed/produced. In this approach a *Workproduct* can be specialized into four kinds of artifacts: a *UML model*, transformed/generated during the process enactment; a *Transformation*, to any kind of transformation; a *Transformation Rule*, which contains the rules for automatic model transformation and code generation; an *Extra Model*, textual specifications or supplementary notation necessary for the project; and a *Profile*, which gives additional and specialized semantics for system modeling according to a specific application domain or platform. Tasks are also performed in a specific tool (*Modeling Tool* or *Transformation Tool*) which can be modeling tools (e.g. magic draw) or transformation tools (e.g. ATL engine).

[8]Eclipse Process Framework—http://www.eclipse.org/epf/.

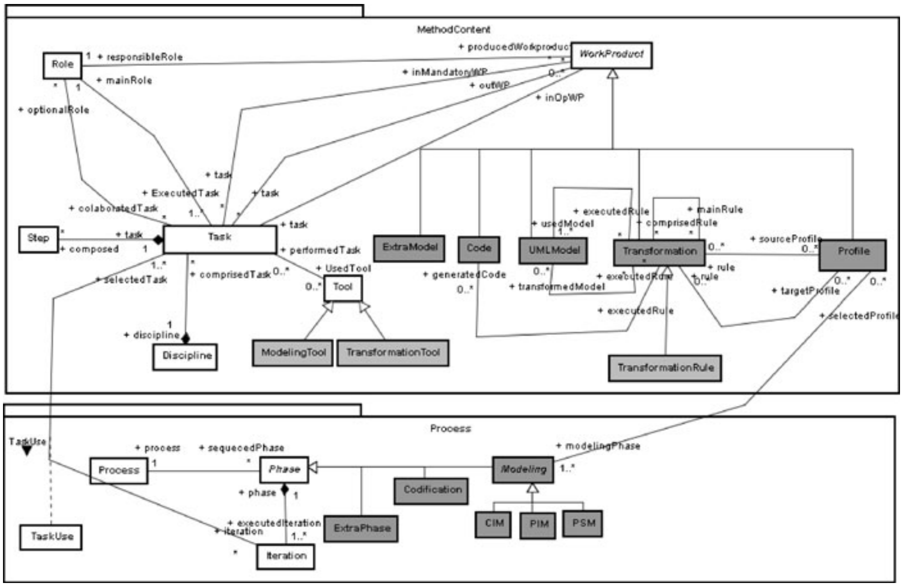


Fig. 4 Core metamodel (adapted from Maciel et al. 2009)

Based on static definitions many processes are modeled using metamodel dynamic concepts. A *Process* may comprise many *Phases* specialized in modeling *CIM*, *PIM*, *PSM* (OMG 2003) and also in *Codification*. Moreover, an *ExtraPhase* can be specified representing an additional stage apart from modeling and codification. The modeling phases can be associated with profiles to support modeling tasks. Each *Phase* can contain one or more *Iterations* that specify the *TaskUses* necessary to carry out a task.

According to this metamodel, in MoDErNE a MDD process can be diagrammatically specified by the construction of three kinds of UML diagrams (class, use case and activity diagrams), following the concepts of the metamodel. It is therefore possible to model class “Software Architecture Definiton” or “Service Interface Design” and associate to the PIM phase stereotype, according to the MDD process characteristics. Furthermore, a *Transformation Rule* artifact, modeled as a class named “UseCaseToClass”, which maps use case elements into class, can be associated to another element stereotyped as *Transformation* to form, for example, a transformation chain.

3.1.2 Metamodel for testing

A specific metamodel for testing processes has been built to enable the explicit definition of process elements concerning model-driven testing within an MDD process. It is based on several concepts of an IEEE Standard (IEEE 2008). The IEEE Standard was chosen because it proposes a complete testing process and documentation, providing test activities for each part of the software life cycle.

The metamodel for Testing is illustrated in Fig. 5. Some meta-classes were specialized from the Core Metamodel (Fig. 4).They include concepts which can be treated

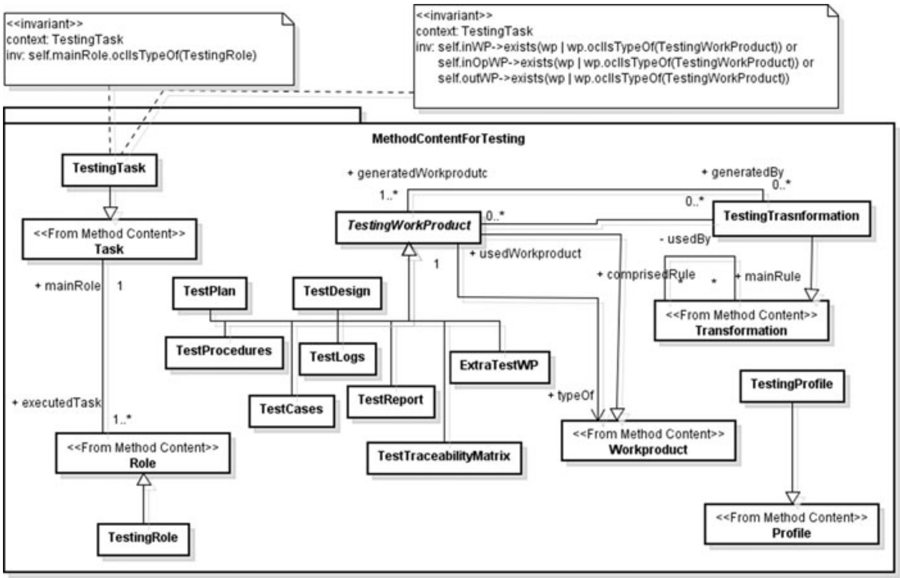

<<invariant>>
context: TestingTask
inv: self.mainRole.oclIsTypeOf(TestingRole)
<<invariant>>
context: TestingTask
inv: self.inWP->exists(wp | wp.oclIsTypeOf(TestingWorkProduct)) or
self.inOpWP->exists(wp | wp.oclIsTypeOf(TestingWorkProduct)) or
self.outWP->exists(wp | wp.oclIsTypeOf(TestingWorkProduct))
MethodContentForTesting
TestingTask
<<From Method Content>> Task
+ mainRole 1
+ executedTask 1..*
<<From Method Content>> Role
TestingRole
+ generatedWorkprodutc 1..*
TestingWorkProduct
0..*
+ usedWorkproduct
1
+ generatedBy 0..*
0..*
TestingTrasnformation
- usedBy
+ compriseRule *
+ mainRule
<<From Method Content>> Transformation
TestPlan
TestDesign
TestProcedures
TestLogs
ExtraTestWP
TestCases
TestReport
TestTraceabilityMatrix
+ typeOf
<<From Method Content>> Workproduct
TestingProfile
<<From Method Content>> Profile

Fig. 5 Metamodel for testing (adapted from Maciel et al. 2011)

specifically for testing specification, and also some of them can be handled in model transformations. *Task*, *Role* and *Workproduct* are concepts that belong to the *Method Content* package of the Core metamodel, making the connections between both metamodels. The associations and/or generalizations with the concepts of the *MethodContent* package complement the comprehension of the metamodel for testing.

A *TestingTask* is a kind of *Task* which is constrained by the OCL (Object Constraint Language) rules described at the top of Fig. 5. The leftmost OCL rule indicates that a *TestingTask* must have, as input or output, at least a *Workproduct* which is a *TestingWorkProduct*. The rightmost OCL rule requires that the main *Role*, responsible for a *TestingTask* execution, must be a *TestingRole*, which is a role in the process especially involved in testing activities.

A *TestingWorkProduct*, an artifact for testing, can be generated or consumed by a *TestingTask*. The *TestPlans*, for instance, are the documents that contain the planning information related to the test, such as the schedule of its implementation and execution, and are usually developed in the early phases of the process. The *TestCases* represent the code that is going to execute the system in order to find errors. *TestDesign* and *TestProcedure* represent the structural and behavioral aspects of the test respectively and adapted to MDT context, there are testing model artifacts. *TestReport* and *TestLog* are workproducts produced after the execution of the tests to document the results. *TestTraceabilityMatrix* track what requirements are being tested by a test case and *ExtraTestWP* represents extra documentation occasionally needed by the test activities.

The *Transformation* class, of the Core metamodel, was extended to the *TestingTransformation*. This allows exclusive treatment when transformations are carried out for MDT. Transformation rules for MDT should generate at least a *TestingWorkProd-*

uct, using a *TestingProfile* or not, which can define specific modeling notation for modeling tests, such as U2TP (UML 2.0 Testing Profile) (OMG 2005).

As we explained before, a *Process* package can use more than one *MethodContent* for distinct purposes. It is important to note that the test metamodel was modeled as a SPEM *MethodContent* which can be reused in several and different software processes.

3.1.3 Metamodel for process management

The metamodel for Project Management is based on the PMBOK (Project Management Body of Knowledge) (PMI 2008). A slice of this metamodel is illustrated in Fig. 6. PMBOK was chosen because this approach presents several disciplines for the process management life cycle. We tried to follow the well-established concepts of PMBOK associating and extending the concepts of the Core metamodel (Fig. 4).

Like the other metamodels, the *ManagementDiscipline* is the specialization of *Discipline* from *MethodContent*. It is divided into nine kinds of disciplines corresponding to the nine knowledge areas of PMBOK, such as *Project Scope Management*, *Project Time Management* and *Project Cost Management*. Due to the lack of space here, we only show three of the nine disciplines in Fig. 6.

A *ManagementTask* is a specialization of *Task* from *MethodContent*. Management tasks are also divided into nine kinds of tasks to represent the management tasks corresponding to the so called 42 "processes" in the knowledge areas in PMBOK. Similarly, only three of the nine tasks are illustrated in the metamodel slice shown in Fig. 6. Finally, a *Stakeholder* (specialization of *Role* from *MethodContent*) is responsible for one or more *ManagementTasks*. The idea is similar to the relationship between *TestingRole* and *TestingTask* from the metamodel of Fig. 5. The stakeholders

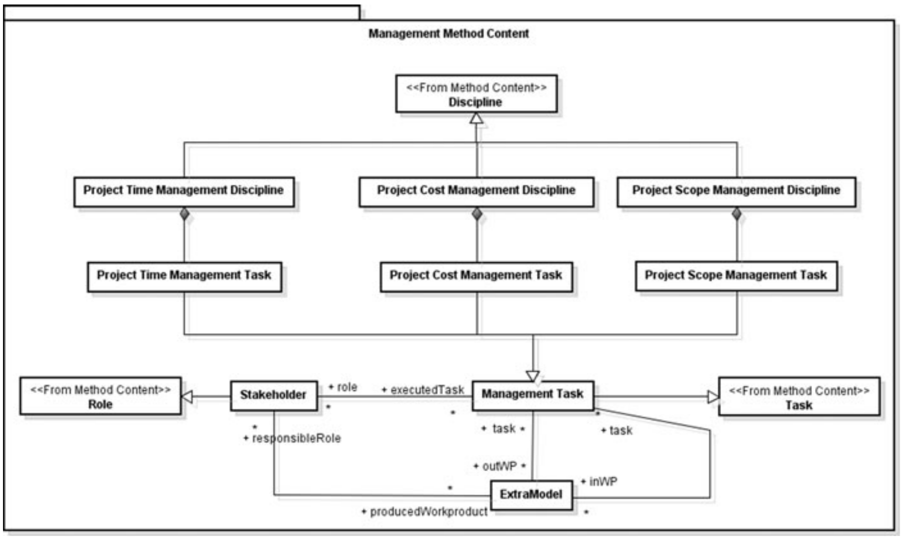

Management Method Content
<<From Method Content>> Discipline
Project Time Management Discipline
Project Cost Management Discipline
Project Scope Management Discipline
Project Time Management Task
Project Cost Management Task
Project Scope Management Task
<<From Method Content>> Role
Stakeholder
+ role
+ executedTask
Management Task
<<From Method Content>> Task
+ responsibleRole
+ task
+ task
+ outWP
+ inWP
ExtraModel
+ producedWorkproduct

Fig. 6 Management metamodel (slice)

are consequently responsible for generating the artifacts (*ExtraModel*) which are the inputs and outputs of each *ManagementTask*. The project management can be defined through management tasks across an MDD process or it can be defined separately as a *ManagementProcess* itself.

3.2 MoDErNE's architecture

Based on the previously explained metamodels, MoDErNE has been developed to support MDD development testing and management process modeling and enactment. MoDErNE's general architecture is shown in Fig. 7. The tool contains two main modules with complementary resources, namely the *ProcessEditor* and the *ProcessExecutor* components. The *ProcessEditor* is responsible for the process specification following the conceptual metamodels. The *ProcessExecutor* supports the enactment of the processes previously specified in the *ProcessEditor*.

The two main components communicate with each other through a repository which is a relational database (MySQL database). Either the process models, created in the *ProcessEditor*, or artifacts such as the application models created during the development lifecycle in the *ProcessExecutor* remain in this repository.

A client-server architecture style is used to provide collaborative enactment of the process. A RMI (*Remote Method Invocation*) based component (*Server*) communicates various instances of the *ProcessExecutor* to the same repository. This repository is accessed and controlled by the *PersistenceControl* component. By using *ProcessExecutor*, the developers can persist models, code and other artifacts and make them accessible to other team members. This collaborative environment plays an important role as software team distribution and cooperation is one of the key functionalities that should be provided by PSEEs.

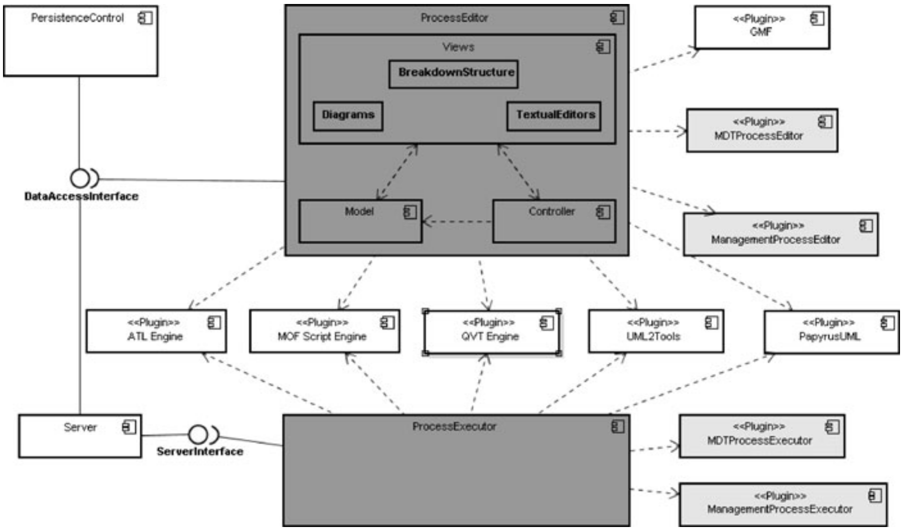


Fig. 7 MoDErNE's general architecture (adapted from Gomes et al. 2011)

The main modules are implemented as RCP (*Rich Client Platform*) products under the Eclipse platform. It is thus possible to use Eclipse's graphic widgets as well as some of the several plug-ins developed for this platform, such as: ATL and QVT for model-to-model transformation rule editing and execution; MOFScript, for the editing and execution of model-to-text transformation rules; GMF (*Graphical Modeling Framework*)[9] for the customized diagram graphic editor creation and generation; EMF (*Eclipse Modeling Framework*), for modeling and automatic code generation of process models, which are based in the metamodels; and UML2Tools[10] and PapyrusUML,[11] which are both plug-ins for the modeling of profiles and other UML artifacts. As an Eclipse RCP product, MoDErNE also provides extension points that make it possible to expand support to the process enactment, by integrating other useful Eclipse plug-ins to the environment.

Based on the metamodels and on the specified process, the tool identifies the element type and invocates the correct plugin to support an activity. MDT support as well as project management support is provided as an Eclipse plug-in. Therefore, the user can choose whether to have testing and management support or not. There are four plug-ins (Fig. 7—lighter grayscale) in MoDErNE: *MDTProcessEditor* and *ManagementProcessEditor* extend the *ProcessEditor* functionalities, enabling the modeling and editing of testing and management processes respectively; and *MDTProcessExecutor* and *ManagementProcessExecutor*, extend the *ProcessExecutor* functionalities in order to support the enactment of the MDT and management processes previously created in the *ProcessEditor*.

The *ProcessEditor* component uses a MVC (*Model View Controller*) pattern. The Editor provides three different means of specifying and visualizing process elements: (i) *BreakdownStructure* is a view that represents the process in a hierarchical structure (ii) *Diagrams* is a view in which the process is represented by UML diagrams (classes, use cases or activities) (iii) *TextualEditors* consists of a set of editors that extend the editors provided by the Eclipse Platform and enable process data editing. These views reflect information about the processes that are encapsulated in the *Model* subcomponent. Changes made in a process class diagram, for example, are thus propagated to the other views. The *Controller* subcomponent coordinates the communication between the *Model* and *Views* so that different views correspond to a single *Model* element.

The following sections detail the functionalities provided by the two main modules of the environment: the *ProcessEditor* and the *ProcessExecutor*.

3.3 MoDErNE modules

The following sections detail each one of these modules using the *Integrated Process* as an example. This process uses the specification of the MDA process for middleware specific services proposed in Maciel et al. (2006) and MDT process proposed in Maciel et al. (2011). This MDA process goal is to develop middleware services

[9]GMF—-www.eclipse.org/gmf/.

[10]UML2Tools—http://www.eclipse.org/modeling/mdt/?project=uml2tools.

[11]PapyrusUML—http://www.papyrusuml.org.

in Java EE or CORba CCM platforms, based on application functional requirements. Initially, this process was defined in an ad hoc manner and then specified using MoDErNE metamodels. The MDT process generates test cases for JUnit platform from UML models stereotyped using U2TP profile.

The *Integrated Process* specification uses two reusable libraries (SPEM method content) named *middleware service develop*, with only strategies for middleware services development, an instance of the Core Metamodel (Sect. 3.1.1); and *model driven test*, with only strategies for software validation and test generation, based on the MDT metamodel (Sect. 3.1.2). Its specification is stored in the repository to be used by the Process Executor in several other software development projects. The Process Executor section illustrates the *Integrated Process* enactment in a case study that develops a bank application.

To perform the *Integrated Process* specification and enactment without MoDErNE, the software development team had to use some different kinds of tools: (i) for process specification (e.g. EPF), (ii) UML editor diagram according metamodel stereotypes process needs (iii) an engine for model-to-model transformation (iv) model-to-text transformation engine, and (v) for process management. Using MoDErNE, the development team will have, in addition to the features of each tool mentioned above, a repository to store the artifacts that were used and produced in each task available. Moreover, the process specification works as a support for task delegation among project members, as well as a guide for task sequence and workproducts required for each tasks.

3.3.1 MoDErNE process editor

To specify a new process, the first step is to select existing method content or define a new method content library with the disciplines, tasks, roles and workproducts that will be used in the process definition. After this, processes can be specified. A process can use all method content elements or only some of them.

Figure 8 shows the method content *Model Driven Test* with elements defined for the *Test Implementation Discipline*. It illustrates how class diagrams can be used to represent the method content elements and their relationships. In the figure there is a *Discipline* called *Test Implementation* which comprises a set of tasks such as *Generate Component Test Cases* and *Execute Component Test*. It also shows the input and output *Workproducts* which are used and generated by each task. For example, *Generate Component Test Cases* task uses as input *Test Design* and *TestModelToTestCode* workproducts and, as output *Component Test Case*. The available class editor shows a tool pallet with the CORE and Testing metamodel concepts to be used in the process edition. It is important to note that each element is stereotyped according to these metamodels. This allows us to give specific treatment to such elements. For instance, the tool can recognize that the *TestModelToTestCode* artifact is a transformation rule (as it is stereotyped as *TestingTransformation*) and then provides users with the integrated ATL environments for rule creation and edition. It is possible to use transformations developed outside MoDErNE tool, if they were developed in ATL, QVT of MOF-Script, by importing their transformation code as a *Transformation* artifact.

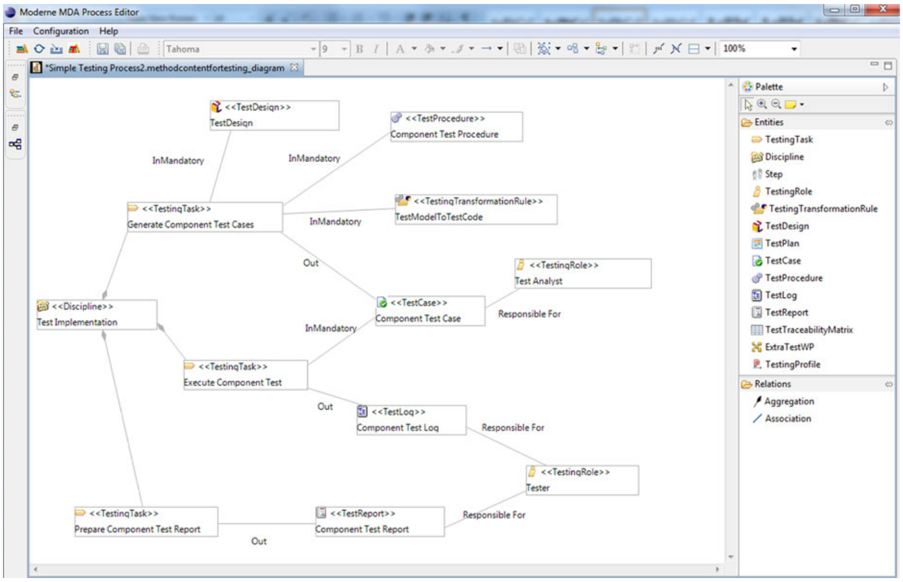

Moderne MDA Process Editor
File Configuration Help
*Simple Testing Process2.methodcontentfortesting_diagram
<<TestDesign>> TestDesign
<<TestProcedure>> Component Test Procedure
<<TestingTask>> Generate Component Test Cases
<<TestingTransformationRule>> TestModelToTestCode
<<TestingRole>> Test Analyst
<<Discipline>> Test Implementation
<<TestCase>> Component Test Case
<<TestingTask>> Execute Component Test
<<TestLog>> Component Test Log
<<TestingRole>> Tester
<<TestingTask>> Prepare Component Test Report
<<TestReport>> Component Test Report
InMandatory
Out
Responsible For
Palette
Entities
TestingTask
Discipline
Step
TestingRole
TestingTransformationRule
TestDesign
TestPlan
TestCase
TestProcedure
TestLog
TestReport
TestTraceabilityMatrix
ExtraTestWP
TestingProfile
Relations
Aggregation
Association

Fig. 8 Method Content class diagram example

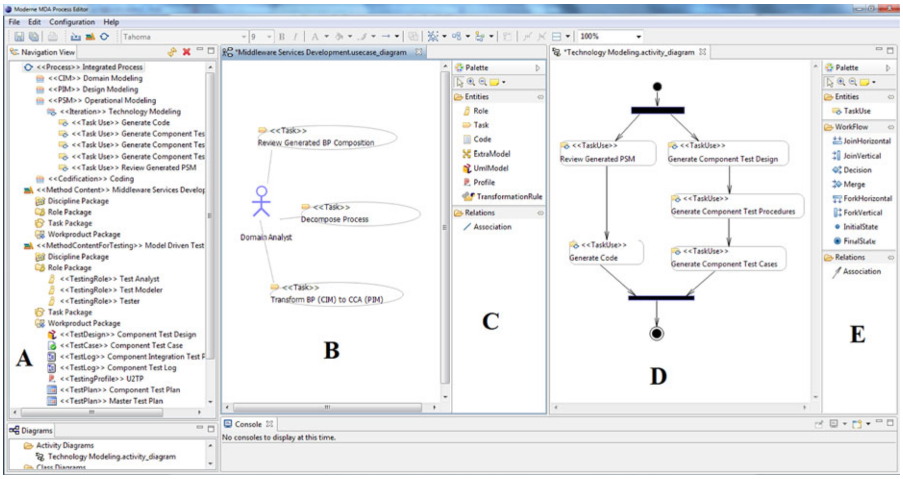

Moderne MDA Process Editor
File Edit Configuration Help
Navigation View
<<Process>> Integrated Process
<<CIM>> Domain Modeling
<<PIM>> Design Modeling
<<PSM>> Operational Modeling
<<Iteration>> Technology Modeling
<<Codification>> Coding
<<Method Content>> Middleware Services Develop
<<MethodContentForTesting>> Model Driven Test
<<TestingRole>> Test Analyst
<<TestingRole>> Test Modeler
<<TestingRole>> Tester
*Middleware Services Development.usecase_diagram
<<Task>> Review Generated BP Composition
<<Task>> Decompose Process
<<Task>> Transform BP (CIM) to CCA (PIM)
Domain Analyst
*Technology Modeling.activity_diagram
<<TaskUse>> Review Generated PSM
<<TaskUse>> Generate Component Test Design
<<TaskUse>> Generate Component Test Procedures
<<TaskUse>> Generate Code
<<TaskUse>> Generate Component Test Cases
Palette
Entities
Relations
WorkFlow
Console
No consoles to display at this time.
Diagrams
A
B
C
D
E

Fig. 9 Process Editor

Figure 9 illustrates the Process Editor screen divided into four sections. Pane box (A) presents the process being edited—the *Integrated Process*—, its breakdown structure and elements (*tasks*, *roles*, *workproducts*, etc.). In pane (A) we can see two different *MethodContents*: the *Middleware Specific ServicesMethodContent*, which has elements concerning the development process, and the *Model Driven TestMethodContentForTesting*, which comprises the elements regarding the testing process, such as testing roles (*Test Analyst*, *Test Modeler* and *Tester*) and testing workproducts (*Com-*

ponent Test Design, *Component Test Case*, etc.), as well as certain artifacts related to it, such as the UML 2.0 Testing profile (*U2TP*). Panes (B) and (D) correspond to visual modeling areas in which the user can create and edit process elements through UML diagrams. Pane (B) illustrates a use case diagram, used to assign responsibilities to the various process roles. In the example shown in Pane (B), a *Domain Analyst* is responsible for the *Review Generated BP Composition*, *Decompose Process* and *Transform BP* (*CIM*) *to CCA* (*PIM*) process tasks. Pane (D), on the other hand, contains an activity diagram, which represents the activity flow of one process iteration. This represents the sequence of activities of the *Technology Modeling* iteration required to generate code from the PSM Model and the application unit test cases, using ATL Language. It contains two main flows: one comprises *Review Generated PSM* and *Generate Code* tasks regarding the development process; and the other contains *Generate Component Test Design*, *Generate Component Test Procedures* and *Generate Component Test Cases* tasks, which are testing activities. Panes (C) and (E) contain option palettes which support the creation of the process models, instances of the extended SPEM metamodels from Sect. 3.1.

3.3.2 MoDErNE process executor

The first activity to be executed when initiating a new project is to select a process from the repository. The entire environment is configured according to the process definition: phases, iterations and tasks are organized in a hierarchical structure ready to be used. Figure 10 illustrates an Executor screen for the bank application development project. It is divided into two panes. Pane (A) contains the process previously specified in the Editor (Figs. 8 and 9) and selected for the project. In this area, the project manager can assign roles to other users enabling access control to the tasks. This will be used to control the collaborative development of the project.

Although all the tasks of the specified process can be visualized by each development team member, task execution is individual. Role definition is used to manage the collaborative project controlling access to each task by each specific role. Different people may work on the same project at the same time but on different tasks.

Furthermore, the tasks have an icon that indicates whether they are being performed (), have already been finished (✓), have not yet started but are ready to start (), or have not started and depend on another task that has not yet finished (⊘).

Pane (B) consists of the modeling environment in which the developer can create the application models. According to the workproduct stereotype specified for each task in the process specification, MoDErNE associates and makes the necessary tool available to support the task performing. For example, in Fig. 10(B), the user is performing the *GenerateComponentTestCase* task. This task was specified to produce a workproduct stereotyped as a *TestDesign* (as can be seen in Fig. 8), which is a *UMLModel*. Therefore, MoDErNE opens a UML class editor for the *Transaction-TestCase* class diagram modeling. The UML profiles specified in the process in the Editor are automatically available to use in the Executor and its stereotypes can be applied by the developer while the diagrams are being created. U2TP (UML 2.0 *Testing Profile*), a profile for system testing, is available in the environment and whenever a

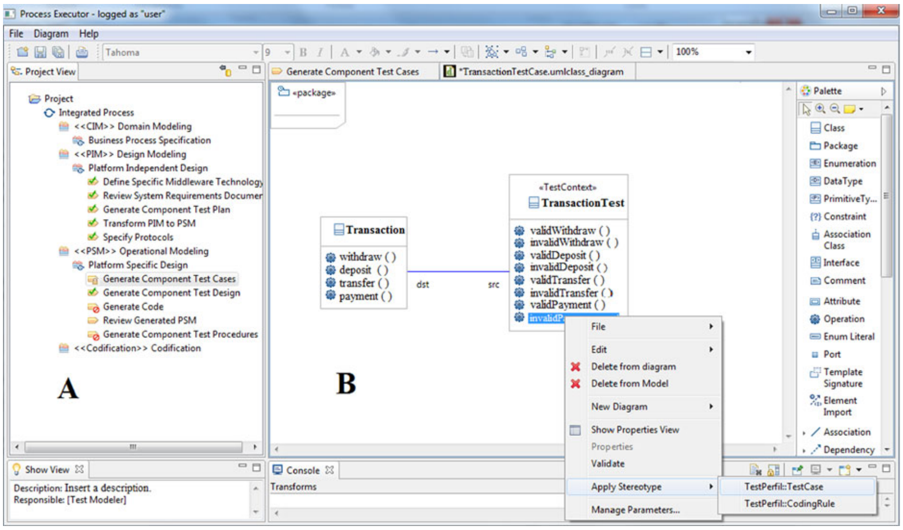


Fig. 10 Process Executor

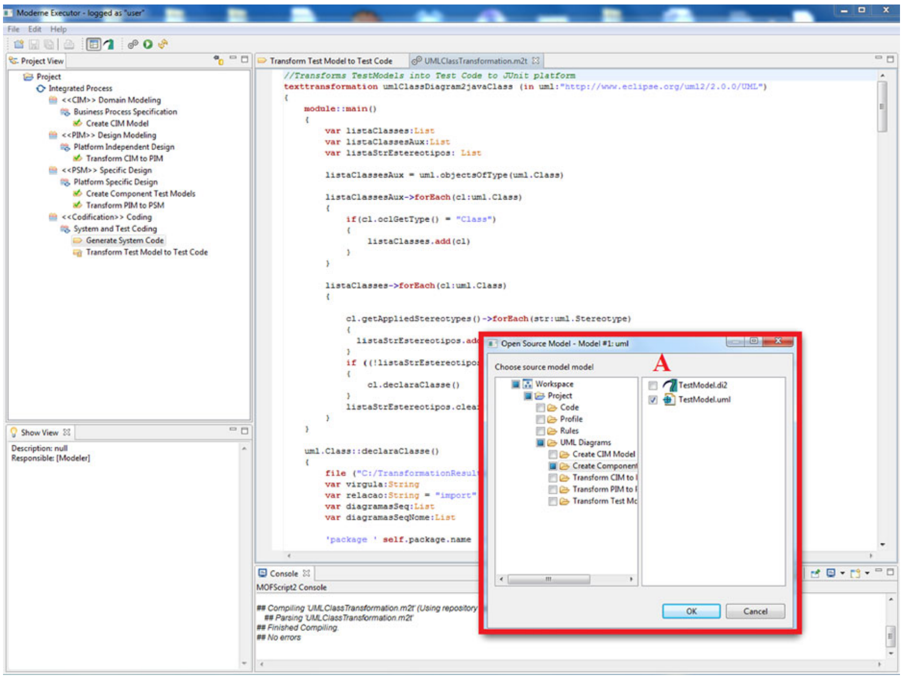
testing task is performed it is enabled, allowing users to annotate their models with the stereotypes of the profile. This feature is possible because an *UMLModel* workproduct can be associated to a *Profile* workproduct according to the Core metamodel (see Fig. 10 popup).

In the Fig. 10 example, unit test cases are being modeled and the user can apply U2TP defined stereotypes. These test case models can later be transformed into JUnit test case codes for the JUnit platform.

Figure 11 illustrates MoDErNE's support for model transformation enactment. Whenever the performed task were associated to a workproduct stereotyped as a *Transformation*, the correct transformation engine is automatically shown to the user. The example in Fig. 11 exemplifies a MOFScript transformation rule (*UMLClassTransformation.m2t*) to generate test code from test models. The user selects the previously created input model and then executes the transformation generating the testing code.

4 Case studies

This section presents our work on the evaluation of our environment through case studies that have been carried out. Evaluating software process specification and enactment is not an easy task due to the complexity of usage scenarios. In general, many people enact on processes but few of them specify or model a software process. We have carried out different case studies over recent years in different contexts and scenarios using some GQM-based (Goal-Question and Metric) assessment techniques, except in the first one. For the case studies we establish a main goal, pose some questions, draw up a scale for questions, answers and metrics concerning the goal. Each



Description: null
Responsible: [Modeler]

```
//Transforms TestModels into Test Code to JUnit platform
texttransformation umlClassDiagram2javaClass (in uml:"http://www.eclipse.org/uml2/2.0.0/UML")
{
    module::main()
    {
        var listaClasses:List
        var listaClassesAux:List
        var listaStrEstereotipos: List

        listaClassesAux = uml.objectsOfType(uml.Class)

        listaClassesAux->forEach(cl:uml.Class)
        {
            if(cl.oclGetType() = "Class")
            {
                listaClasses.add(cl)
            }
        }

        listaClasses->forEach(cl:uml.Class)
        {

            cl.getAppliedStereotypes()->forEach(str:uml.Stereotype)
            {
              listaStrEstereotipos.add
            }
            if ((!listaStrEstereotipo
            {
                cl.declaraClasse()
            }
            listaStrEstereotipos.clea
        }
    }

    uml.Class::declaraClasse()
    {
        file ("C:/TransformationResult
        var virgula:String
        var relacao:String = "import"
        var diagramasSeq:List
        var diagramasSeqNome:List

        'package ' self.package.name
```

Choose source model model

```
## Compiling 'UMLClassTransformation.m2t' (Using repository
   ## Parsing 'UMLClassTransformation.m2t'
## Finished Compiling
## No errors
```

Fig. 11 MoDErNE's support for model transformations

case study can encompass one more experiment. After each experiment, a questionnaire is applied. Then the answers are recorded and analyzed. After each experiment we analyze the artifacts produced in the process specification and enactment tasks.

Table 1 summarizes the four main case studies we carried out. The first one explored the experience of the specification of an MDD process for the development of specific middleware services, and in this study we did not use GQM-based assessment as it was performed in our research laboratory. The second case study was the specification of an MDA process for web-based applications which was performed together with a company in industry (Sect. 4.2). The third and fourth case studies encompassed different kinds of MDD and MDT scenarios: process specification, understanding a previously specified process and process enactment using the MoDErNE environment (Sect. 4.3). Case Studies for Management process context have not been performed yet as these functionalities are still undergoing validation.

4.1 First and second case studies

As these case studies were detailed in Maciel et al. (2009), this paper gives an overview in order to support our findings about the proposed tool. The first study (SC1-S1), the MDA process for middleware services modeling, aims to verify our approach and tool feasibility. The MDA process goals presented in Maciel et al. (2011) encompass the specification and implementation of portable specific middleware services. The process specification lifecycle comprises four phases: CIM, called Domain

Table 1 Case studies and scenarios

Case Study	Process Specification	Process Enactment	Process Understanding
SC1-S1	√	–	–
SC2- S2	√	–	–
SC3-S3	√	–	–
SC3-S4	–	√	–
SC4-S5	–	–	√
SC4-S6	–	√	√

Model, which corresponds to the context in which the service should be applied including enterprise information viewpoints; PIM, called Design Model, responsible for the computational view including services offered and their operations; PSM, called Operational Model, to add a Technology viewpoint to the specification on a specific platform; and codification. This process is used in several projects in our research laboratory, but it was originally described without any standard language. Tables, illustrations and textual documents were used to represent the process specification. An engine was developed to support the automation of model transformations related to the process (Maciel et al. 2009). However, the difficulties in understanding, reusing and evolving the process structure and behavior across development teams became evident. After using the MoDErNE environment the process elements could be specified through the standard concepts according to SPEM and specialized definitions following our Core metamodel (Fig. 4, Sect. 3). Thus, the process became easier to understand and present to new people interested in the process structure and behavior. The process also became easier to evolve after these modeling, new testing and development tasks were added.

The second case study (SC2-S2) was to model a process from a real company called PRODEB (Data-Processing Company of Bahia State) which was important to assess the applicability of our approach. This study involved the modeling of the PRODEB process for the development of web applications using the MDD approach. PRODEB had been using the AndroMDA tool for a couple of months during the development of a web-based application. However, they encountered limitations related to the tool environment especially because the process definitions (phases, activities, artifacts, roles, transformations etc.) were not specified and documented. Therefore, the process knowledge had not been registered so far. Furthermore, as MDD is an emerging technology not all the professionals were familiar with it. Most of the PRODEB staff did not understand why they had to stereotype UML elements or why they had to elaborate some models, which are necessary activities for the AndroMDA tool. In this context, we worked together with the team of professionals from PRODEB for a couple of months to model the PRODEB process using our approach. At the end of this period, a questionnaire with six questions was applied.

Additionally, in this case study we observed that difficulties in process comprehension, mostly related to the execution sequence of activities, were eliminated. At several moments developers suggested ways to improve the specification of the current MDA process. As the process was designed by the professionals from PRODEB

Table 2 GQM summary for assessing MoDErNE process specification

Goal	Question	Metric
(G1) Verify the applicability of the MoDErNE environment for process specification.	**(Q1.1)** Is it possible to clearly define the sequence of phases?	**(M1.1)** Degree of ease and comprehension to define process phases
	(Q1.2) Is it possible to clearly define the tasks?	**(M1.2)** Degree of ease and comprehension to define process tasks
	(Q1.3) Is it possible to clearly define the roles involved in each task?	**(M1.3)** Degree of ease and comprehension to define process roles and associate them with tasks
	(Q1.4) Is it possible to clearly define the workproducts and their (input/output) associations with the tasks?	**(M1.4)** Degree of ease and comprehension to define workproducts and associate them with tasks
	(Q1.5) Does the MoDErNE tool support process specification properly?	**(M1.5.1)** Degree of effort to put in the process specification using our tool
		(M1.5.2) Time spent on processes specification
		(M1.5.3) Degree of expected effort in the next process specification

could better understand their own work and they used our meetings to discuss new definitions and elements to improve their process. We can therefore conclude that our approach contributed to the comprehension, evolution, reuse and enactment of the MDA processes we worked on.

4.2 Third case study

In the third case study, the goal was to evaluate the applicability of the MoDErNE tool. This case encompasses two different scenarios. The first scenario (SC3-S3) was to verify MoDErNE Process Editor in relation its support for process specification and the second was to verify the MoDErNE Process Executor with regard to its support for process enactment (SC3-S4).

The participants in this study were fourteen graduates working in the industry. Most of them were systems analysts working for software factories and banks.

First of all, we gave them some lectures about MDD concepts, technologies and also provided training on our approach and tool totaling 16 hours divided into 4 days in one week (4 h/day). The questionnaire which was answered individually by each participant was divided in two parts: the first concerning the MoDErNE Process Editor regarding process specification and modeling; and the second regarding the MoDErNE Process Executor.

4.2.1 (SC3-S3) assessing the authoring of an MDD process in MoDErNE editor

In the first scenario from the third case study (SC3-S3) we elaborated a different scenario as our goal was also to evaluate the applicability of the MoDErNE environment

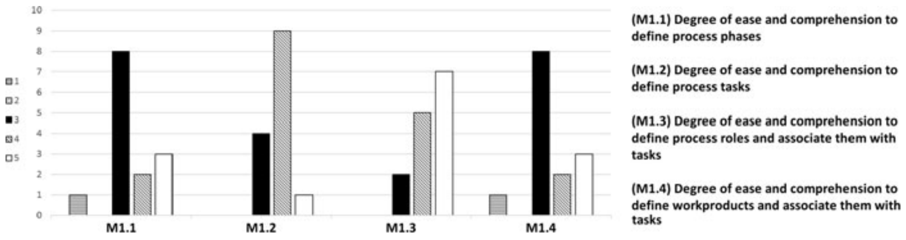


Fig. 12 Answers about the simplicity of defining tasks, roles, workproducts and phases

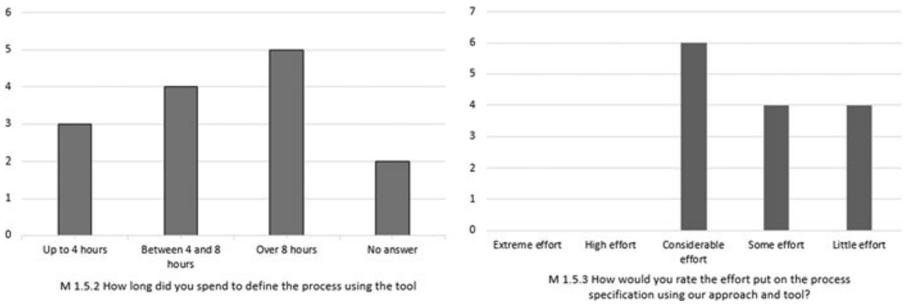


Fig. 13 (**a**) Time and (**b**) Effort spent on process specification

regarding the MDD process specification without our intervention. Then we organized the students into 4 groups and asked them to specify an MDD process from their experience at work using the MoDErNE Process Editor. We gave them a deadline, but the specification time was not restricted. They had all the time they needed to organize themselves in their group to do the job and deliver the process. After the process specification the resulting process models were checked.

After process delivery, we also applied a questionnaire and started analyzing the process specification. The questionnaire had a total of 15 questions related to the (GQM) method for the generation of necessary questions and metrics concerning our initial goals, some of which are detailed in Table 2. A GQM question can be related to one or more questionnaire answers, while the answer for a question is related to a GQM metric for a question. For metric measures, the answers have a scale reference (e.g. some effort, high effort, low effort, etc.). The collected results and our analysis are presented below.

The following figures show some results from questions regarding tool support in the MDD process specification. The charts in Figs. 12, 13 and 14 show some results from the questionnaire. The first one (Fig. 12) presents the answers related to questions about the simplicity of defining *tasks*, *roles*, *workproducts* and *phases*, and is related to questions Q1.1 to Q1.4 from Table 2. The scale for the answer was 5—Easily, 4—Reasonably, 3—Satisfactorily, but with some difficulty, 2—Inadequate, 1—Could not.

Most of the participants' answers demonstrated that it was possible to define all the process elements (Fig. 12), despite many never having worked with the MDD approach, and some of them not having worked with a previously defined process.

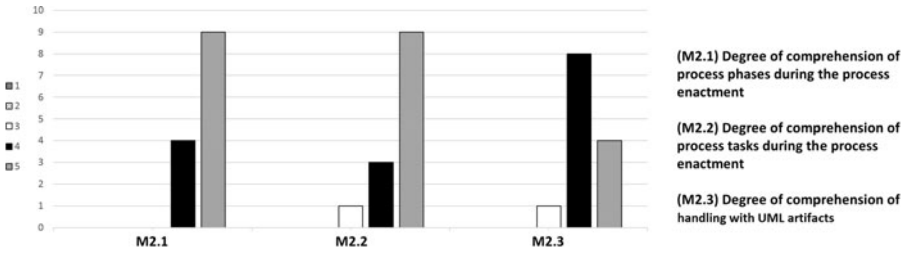


Fig. 14 Answers about phases and task visualization and understanding

Besides the aspects discussed from the results summarized in the above charts and from analyzing the process models resulting from the experiment, it was also possible to observe that all the processes defined by the participants had well-defined modeling (CIM, PIM and PSM) and codification phases, including tasks and steps, roles assignment, associated *workproducts* and also transformation rules. Therefore, it can be concluded that our approach and tool enabled process definition with the expected characteristics of a traditional software process while also adding the peculiarities of an MDD process.

Figure 13 shows the results regarding adequate support for the MDD processes specification aspects in MoDErNE (Table 2, Q1.5, M1.5.1 and M1.5.2). To measure these aspects we asked three questions about the time and effort taken to perform the process specification tasks.

Figure 13(a) indicates that half of the participants spent less than eight hours in their group meetings for the process specification, answered by each group member. Less than a half spent more than eight hours and two participants did not answer the question. Figure 13(b) indicates that more than half of them considered that it took little effort while 42 % participants thought it took considerable effort.

Regarding the degree of expected effort in the next process specification (M1.5.3), most of the participants (93 %) agree that the effort put into the process specification would not be repeated in the future if they used the MoDErNE environment for specifying the new process. That is, the time spent learning the approach and tool would not be repeated. We also should consider that the process definitions in the method content remain available for reuse. This can possibly reduce time and future effort in process specifications.

Some participants had difficulty understanding some process definitions as presented in the tool, however, none of them rated the process comprehension as presented in the tool negatively. The effort put into the process specification is valuable but necessary. Most of them classified such effort as reasonable but not enough to rate it as a negative point. Besides, part of that effort would not be repeated on future occasions when using the tool.

Considering such aspects as: (i) the training time for both MDD approach and tool, (ii) students inexperience with process specification tasks, (iii) the resulting process models were well formed according to the XMI (XML Metadata Interchange) format, we can say that MoDErNE facilitated the MDD process specification task in the experimental scenario.

Table 3 GQM summary for assessing MoDErNE process enactment

(G2) Verify the applicability of the MoDErNE tool for process enactment.	**(Q2.1)** Is it possible to visualize and understand the process phases during the process enactment supported by the MoDErNE tool?	**(M2.1)** Degree of comprehension of process phases during the process enactment
	(Q2.2) Is it possible to visualize and understand the process tasks during the process enactment supported by the MoDErNE tool?	**(M2.2)** Degree of comprehension of process tasks during the process enactment
	(Q2.3) Is it possible to clearly create, edit and visualize the UML artifacts produced during the process?	**(M2.3)** Degree of comprehension of handling with UML artifacts
	(Q2.4) Is it possible to clearly execute the model-to-model transformations during the process enactment?	**(M2.4)** Degree of comprehension of executing model-to-model transformations
	(Q2.5) Is it possible to clearly execute the model-to-code transformations during the process enactment?	**(M2.5)** Degree of comprehension of executing model-to-code transformations
	(Q2.6) Is the process representation provided by the MoDErNE tool during the process enactment easy to understand?	**(M2.6)** Degree of comprehension of the process representation provided by the MoDErNE tool during process enactment
	(Q2.7) What is your general impression of the MoDErNE tool for the process enactment?	**(M2.7)** Recommendation degree of the MoDErNE tool for supporting the processes enactment.

4.2.2 *(SC3-S4) assessing a MDD process enactment in the MoDErNE process executor*

In the fourth scenario in the third case study (SC3-S4) our goal was to evaluate the applicability of the MoDErNE environment regarding the MDD process enactment. A key point in the MDD process enactment is to produce models according the meta-model stereotypes and execute the transformations properly. Developers must know the exact task sequence to generate correct models and code. The case study goal was to observe MoDErNE's support for these tasks.

The participants were the same as in the first experiment of the third case study. In this experiment each one had to enact several tasks (modeling and transformation) from the MDA process for middleware service modeling to implement services for a school library management application. These services had to have operations to support book loans, namely, borrow a book, return a book, apply a fine, include, delete and update a book. Like in the scenario described in the previous section, the questionnaire had a total of 15 questions. These questions are related to GQM method for the generation of necessary questions and metrics concerning our goals, of which eight are detailed in Table 3. For the first six questions we used the following scale to rate the questionnaires answers: 5—Easily, 4—Reasonably, 3—Satisfactorily, but with some difficulty, 2—Inadequate, 1—Could not.

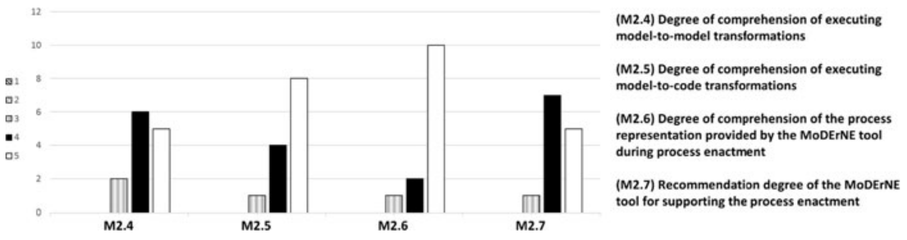


Fig. 15 Answers about transformation tasks, process representation and general impressions

Figure 14 shows the participants' answers about process phases and task visualization and understanding (Q2.1to Q2.3). Most answered that it was easy and reasonable. They also answered that it was easy and reasonably easy to perform modeling and transformation tasks during the process enactment (Fig. 15, Q2.4 through Q2.5). After process enactment, the artifacts (models and code) were checked. One participant failed to generate the models correctly and therefore it was not possible to generate service code.

Q2.6 (Fig. 15) shows the answers about the MoDErNE process representation. As explained in Sect. 3, process elements can be visualized through UML diagrams or breakdown structures. While UML diagrams give more details about tasks and artifacts, it is easier to find and access process elements in a breakdown structure.

The last question, regarding their general impression of MoDErNE (Q2.7 of Fig. 15) we used a different scale: 5—Extremely positive, 4—Positive with few restrictions, 3—Positive, but with important restrictions, 2—Negative. A few positive points I could find, 1—Extremely negative. I did not see a positive side. The majority evaluated the MoDErNE tool support for process enactment extremely positively or positively with some restrictions. Some open answers show that there were some bugs in the tool presented.

Considering that the participants in the experiment were new to the model-driven approach and the service models and the code were correct, we can say that MoDErNE facilitated the MDA process enactment tasks in this case study scenario.

4.3 Fourth case study

In this case study we had the largest number of participants: twenty-nine. We set up a scenario to observe the tool support for a process that used both MDD and MDT techniques. The case study encompassed two scenarios: (i) one to understand a process previously specified in the MoDErNE Editor and (ii) the second to enact this process in the MoDErNE Executor.

The process used in this case study was an abbreviated version of the MDA process for middleware service modeling in which new tasks to test the service were included. The new resulting process contained only modeling, transformation and testing tasks. The specified process contained eight tasks for modeling, implementation and testing of middleware services, 4 of which were transformation tasks (CIM-PIM, PIM-PSM, PSM-code, Test Model → Junit). In this case, the domain being modeled was for a banking application with operations for withdrawal, deposit and bank balances.

Table 4 GQM summary for assessing MoDErNE process specification

Goal	Question	Metric
(G3) Verify the understanding of a previously specified process in the MoDErNE tool	**(Q3.1)** Is it possible to clearly identify the process phases, task and roles?	**(M3.1)** Degree of eases and comprehension regarding process elements
	(Q3.2) Is it possible to clearly identify the artifacts input and output?	**(M3.2)** Degree of ease and comprehension regarding process artifacts
	(Q3.3) Is it possible to clearly identify the transformation tasks?	**(M3.3)** Degree of ease and comprehension regarding transformation tasks
	(Q3.4) Does MoDErNE Editor tool support MDD and MDT process specification properly?	**(M3.4.1)** Degree of ease and comprehension regarding the testing role assessing
		(M3.4.2) Degree of ease and comprehension regarding the testing and development artifacts
		(M3.4.3) Degree of ease and comprehension regarding the testing and development tasks and their sequences
		(M3.4.4) Degree of MoDErNE Editor support regarding the understanding of a process specification.

The case study was divided into four moments. Initially a one and a half hour lecture was given highlighting MoDErNE's features, showing examples of use. Then, they had to use the MoDErNE Process Editor, then the MoDErNE Process Executor and finally they had to answer the questionnaire.

The questionnaire was divided into three parts with a total of 25 questions. The first part with five questions was designed to identify the participants' experience and professional profiles. The other questions were about the metrics established for the experiment using GQM. The scale for the answer was 5—Easily, 4—Reasonably, 3—Satisfactorily, but with some difficulty, 2—Inadequate, 1—Could not.

The participants were students and staff invited from two different universities: the Federal University of Bahia (UFBA) and the Federal University of Campina Grande (UFCG). Regarding the participants' profile, sixty percent described themselves as systems analysts with experience from four to six years. Eighty percent already knew the MDD approach, but had never used it in practice.

4.3.1 (SC4-S5) understanding a process specification through MoDErNE process editor

In the first scenario in the fourth case study (SC4-s5), participants browsed through the process specification in the MoDErNE Editor in order to understand the process elements and task that they would perform later. Soon after the completion of each

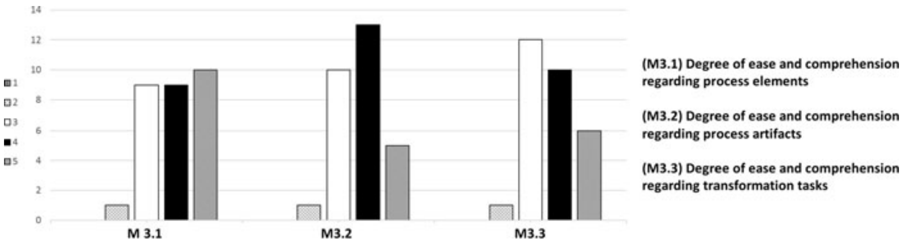


Fig. 16 Answers about process element comprehension

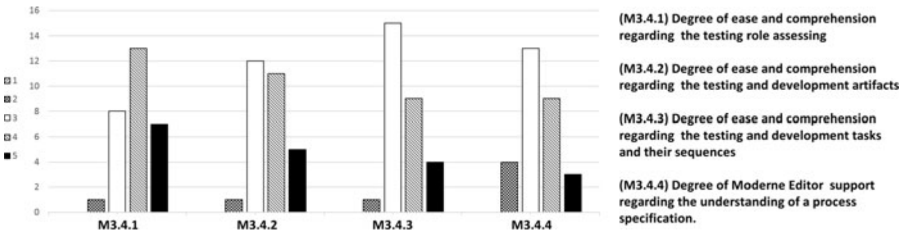


Fig. 17 Answers about implement and testing process elements comprehension

experiment, the participants answered a questionnaire available on the web. Table 4 shows the questions and corresponding metrics.

Figure 16 shows the answers to questions Q3.1 to Q3.3 of Table 4. Although it was the participants first contact with MoDErNE, they answered that they did not have difficulty identifying the phase, tasks, roles, workproducts and transformation rule process elements. As the process mixes both implementation and testing tasks we wanted to observe if the participants could distinguish the testing tasks from the others. Figure 17 shows the answers about these aspects (Q3.4 question 4), most answered that this distinction could be made.

From the answers it can be said that the MoDErNE Editor made the understanding of the implementation and testing activities in this case study scenario possible. While they were browsing the process specification some doubts arose, for example, where to find the button to generate some workflow task diagram. This observation and other similar ones indicate that we have to improve our graphic interface in order to make it more user-friendly.

4.3.2 (SC4-S6) assessing testing and development software process tasks through MoDErNE executor

In this scenario we wanted to observe if, after the process description and brief contact, the participants would be able to perform the assigned tasks. First the participants should create the CIM model of the application, which contains the withdrawal business process representation. Then, they should execute the first tasks related to the transformation chain, making CIM to PIM and PIM to PSM transformations. On completion of the PIM to PSM transformation, the participants had to execute the first testing activity which was to create models stereotyped with U2TP to represent the

Table 5 GQM summary for assessing MoDErNE Process Enactment

Goal	Question	Metric
(G4) Verify the applicability of the MoDErNE Executor module for process enactment for integrated process.	**(Q4.1)** Is it possible to follow the task sequence?	**(M4.1)** Degree of ease and comprehension regarding task sequence
	(Q4.2) Is it possible to perform all the process tasks?	**(M4.2)** Degree of ease to complete process task
	(Q4.3) Is it possible to clearly visualize the transformation task input and output artifacts?	**(M4.3)** Degree of ease and comprehension regarding transformation task artifacts
	(Q4.4) Does MoDErNE environment support MDD and MDT process enactment properly?	**(M4.4.1)** impression for process enactment facilitation **(M4.4.2)** Possibility of carrying out the process without the tool.

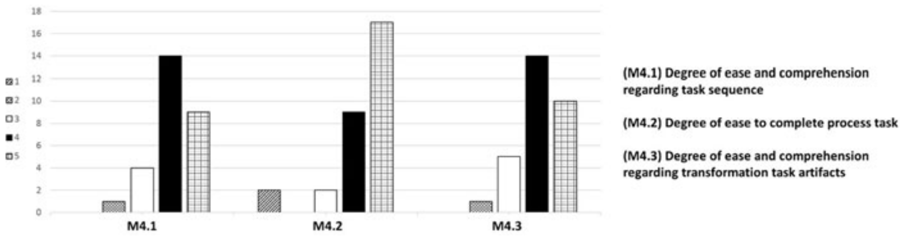


Fig. 18 Answers about the processes task enactment

unit test cases, having the PSM model as input. This test case generates the application's unit test case codes for the JUnit platform by a MOFScript transformation rule. The last task was to generate the application code by applying another MOFScript transformation to the PSM model which had previously been generated.

In the text we did not detail the MDT technique to adopt in the processes. The participants performed the modeling tasks, model to model and model to code transformation tasks without our intervention. However, to perform the transformation testing task a template of a model test artifact was given to the participants. They had to complete the model with U2TP stereotypes and then had to perform the transformations to generate the test cases for the JUnit platform. After this, they had to answer the questionnaire presented in Table 5.

Figure 18 shows the answer concerning question Q4.1 to Q4.3 from Table 5. Most participants could follow the task sequence and performed all the tasks and visualized the artifacts associated to the transformation tasks. Two participants could not perform all the tasks. One of them reported that they could not perform the testing tasks. Another could not finish the process specification browsing activity on time, therefore he did not perform any task. During the process task enactment, some questions were asked about the correctness of the models. Participants were a little insecure about performing transformation tasks without being sure that their models were correct. The Modeling task has some tool support for UML profile stereotype application. If a UML diagram has an associated profile, by clicking the right button on the mouse one stereotype from a list can be chosen that shows only the stereotype that could be applied to each UML diagram element. However, developers wanted to check not only that the diagram was "syntactically correct" but also about its semantic correctness. At the time they were informed that they could see this semantic correctness after the transformation task, if the output model or code was generated correctly. However, considering a longer transformation chain, an error in a stereotype application could become hard to find.

Regarding their impressions about the process enactment facilitation (M4.4.1) most of the participants (87 %) agree that MoDErNE tool makes the process execution easier and 96.7 % of them agree that they would not be able to execute the process without MoDErNE's help (M4.4.2). It can be said that this tool supported MDD and MDT process enactment properly in the case study scenario.

4.4 Lessons learned and study constraints

The MDD approach has been used in our research laboratory in various projects (Bispo et al. 2010; Maciel et al. 2005; Magalhães et al. 2011). Initially we specified a process in a completely ad-hoc way, using natural language. After searching, trying to use and adapt some tools for our process transformation without success, an engine using the Java language to perform the transformations was developed. Taking into account our experience in the specification and use of MDD processes and studies that indicate the need for adequate processes and tools as a key point to facilitate the use of MDD, MoDErNE was proposed and developed.

The case studies were designed to evaluate specific aspects of model-driven development using MoDErNE: (i) specification and enactment of a process, (ii) integration and use of different strategies for testing and model driven languages and transformation engines, (iii) understanding of a process previously specified.

The first three case studies performed showed positive results in the specification and implementation of model-driven processes while the fourth shows positive results regarding process understanding using MoDErNE tool. Process enactment, evaluated in the third and fourth case studies also achieved positive results. It is important to highlight that the fourth case study proposed scenario is related to the integration of two different strategies in the same process. Despite this, the participants could understand the process, distinguished development and testing activities as well as executing them.

Empirical assessment usually takes into account the amount of data collected from the subjects. However, in the case of an activity of process specification it is difficult to involve a high number of people in the experiments. There are few professionals in organizations involved in this kind of task. In general, many people enact on processes but few specify or model a software process. This observation has already been identified in our previous studies and it is also confirmed here. Empirical assessment in this area facilitates more qualitative analysis than quantitative analysis. Therefore, in the assessment of the process enactment it is easier to collect larger amounts of data, facilitating better quantitative analysis in contrast with assessing process specification. Furthermore, in real scenarios developers perform specific tasks (requirement elicitation, programming, testing, etc.) in a software process, few are involved in different types of tasks. In addition, the MDT and MDD approaches require skills not yet required in traditional process software.

Additionally, metrics to assess the process enactment are more mature in the literature and are also easier to apply in the software industry, such as metrics to evaluate productivity and cost. Nevertheless, metrics to assess process modeling and specification (apart from its enactment) need to be better developed in software engineering.

We should also highlight that the conclusions obtained from our studies are restricted to the particular set of participants. In other words, our analysis regarding the advantages and drawbacks of using our approach and tool may not be directly generalized to other contexts. However, these studies have allowed us to make useful assessments about whether the specification and enactment of MDD processes with a supporting tool is worth studying further. In addition, the studies have also allowed us to make a useful evaluation of the applicability of the MoDErNE tool concerning the specification and enactment of MDD processes, and this can be a starting point for other assessments.

5 Conclusion

This paper presented MoDErNE, an environment that supports software process modeling and enactment based on SPEM 2 concepts for the MDD approach. In MoDErNE, the software process should be specified as an instance of metamodels that describes software process elements that make MDD and MDT concepts explicit. MoDErNE possesses several tools to support different kinds of activities in the MDD and MDT process context. The MDD requires some developer skills, which are not yet widely used in traditional software processes (metamodeling, transformations, etc.). Supporting tools play an important part in establishing MDD use in industry on a wider scale.

In MoDErNE, once a process is specified, it may be used in development application projects. Different methods and techniques can be specified and integrated into a single process description and then customized to its own needs. Using our environment the software process specification has the same conceptual and notational framework. The proposed metamodels become a point of convergence for process integration, specification and enactment. This facilitates the understanding of models both by development teams and by software process automation approaches.

Using MDD and process-centered software engineering concepts in a combined way can help software process systematization and automation. In this scenario, we expect that software processes can be used as a software themselves. As software, it has a specification and automated support for performing tasks. Final users (developers) can run these tasks several times, at different moments to achieve business process goals, which in this context is an application development.

Although the management metamodel is integrated in the environment, its functionalities are under validation and it was not possible to include them in the recent case studies. Several case studies were performed and it was possible to verify the applicability of our environment. As future work, we are planning to conduct case studies with different software processes proposed in the literature, broadening the scope of evaluation for different MDD approaches.

Acknowledgement This work is partially funded by FAPESB, project number 5156/2008, and grant number PES0091/2008.

References

Ambriola, V., Conradi, R., Fuggetta, A.: Assessing process-centered software engineering environments. ACM Trans. Softw. Eng. Methodol., 283–328 (1997)

Arbaoui, S., et al.: A comparative review of process-centered software engineering environments. Ann. Softw. Eng. **14** (2002)

Atkinson, C., Paech, B., Reinhold, J., Sander, T.: Developing and applying component-based model-driven architectures in KobrA. In: Enterprise Distributed Object Computing Conference (EDOC '01) (2001)

Baker, P., Dai, Z., Grabowski, J., Haugen, O., Schieferdecker, I., Williams, C.: Model-Driven Testing: Using UML Testing Profile. Springer, New York (2007)

Bispo, C.P., Maciel, R.S.P., David, J., Ribeiro, I., Conceição, R.: Applying a model-driven process for a collaborative service-oriented architecture. In: Proceedings of the 14th International Conference on Computer Supported Cooperative Work in Design (CSCWD), 2010, vol. 1, Shangai, pp. 378–383 (2010)

Bouquet, F., Grandpierre, C., Legeard, B., Peureux, F.: A test generation solution to automate software testing. In: Proc. of the 3rd International Workshop on Automation of Software Test, pp. 45–48 (2008)

Cass, A.G., Lerner, B.S., Sutton, S.M., McCall, E.K., Wise, A., Osterweil, L.J.: Little-JIL/Juliette: a process definition language and interpreter. In: Proceedings of the 22nd International Conference on Software Engineering, Limerick, Ireland (2000)

da Silva, B.C., Magalhães, A.P., Maciel, R.S.P., Martins, N., Nogueira, L.: Transforms: Um Ambiente de Apoio a Modelagem e Execução de Processos de Software Dirigido por Modelos. In: XXIII Brazilian Symposium on Software Engineering, Tools Session, Fortaleza, Brazil (2009)

Engels, G., Schäfer, W., Balzer, R., Gruhn, V.: Process-centered software engineering environments: academic and industrial perspectives. In: Proceedings of the 23rd International Conference on Software Engineering (ICSE '01), pp. 671–673. IEEE Computer Society, Washington (2001)

France, R., Rumpe, B.: Model-driven development of complex software: a research roadmap. In: Proceedings of Future of Software 2007 (FOSE'07), pp. 35–54 (2007)

Gomes, R., Maciel, R., Silva, B., Silva, F., Magalhães, A.: MoDErNE: model driven process centered software engineering environment. In: Proceedings of CBSoft 2011—II Brazilian Conference on Software: Theory and Practice, Tools Session 2011, São Paulo, Brazil (2011)

Gruhn, V.: Process-centered software engineering environments, a brief history and future challenges. Ann. Softw. Eng. **14**(1–4), 63–382 (2002)

Guelfi, N., et al.: DRIP catalyst: an MDE/MDA method for fault-tolerant distributed software families development. In: OOPSLA Workshop on Best Practices for Model Driven Software Development, Canada (2003)

Hartmann, J., Vieira, M., Axel Ruder, H.: UML-based test generation and execution. White paper, Siemens Corporate Research (2004)

Humprey, W., Kelner, M.: Software modeling: principles of entity process models. SEI Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania (CMU/SEI-89-TR-2) (1989)

Hutchinson, J., Rouncefield, M., Whittle, J.: Model-driven engineering practices in industry. In: Proceedings of the 33rd International Conference on Software Engineering (ICSE '11), Waikiki, Honolulu, pp. 633–642 (2011a)

Hutchinson, J., Whittle, J., Rouncefield, M., Kristoffersen, S.: Empirical assessment of MDE in industry. In: Proceedings of the 33rd International Conference on Software Engineering (ICSE '11), Waikiki, Honolulu, HI, USA, pp. 471–480 (2011b)

IEEE: IEEE Standard for Software and System Test Documentation. IEEE Std 829-2008, IEEE Computer Society (2008)

Javed, A., Strooper, P., Watson, G.: Automated generation of test cases using modeldriven architecture. In: Proc. of the ICSE 2nd International Workshop on Automation of Software Test (AST) (2007)

Koch, N.: Transformation techniques in the model-driven development process of UWE. In: Workshop Proc. of the 6th Intl. Conference on Web Engineering, ICWE '06, Palo Alto, California, vol. 155. ACM, New York (2006)

Lima, A., et al.: Gerência Flexível de Processos de Software com o Ambiente WebAPSEE. In: 20th Brazilian Symp. on Software Eng, Florianópolis, Brasil (2006)

Maciel, R.S.P., Rosa, N.S., Ferraz, C.G.: InterDoc: reference architecture for interoperable services in collaborative writing environments. In: 9th International Conference on Computer Supported Cooperative Work in Design (CSCWD 2005), vol. 1, Coventry, pp. 289–295 (2005)

Maciel, R., Silva, B.C., Mascarenhas, L.A.: An Edoc-based approach for specific middleware services development. In: Proc. 4th Workshop on MBD of Computer Based Systems, Postdam, Germany, p:135–143. IEEE Press, New York (2006)

Maciel, R., Silva, B., Magalhães, A., Rosa, N.: An integrated approach for model driven process modeling and enactment. In: XXIII Software Engineering Brazilian Symp, Fortaleza, Brazil, pp. 104–114 (2009)

Maciel, R., Gomes, R., Silva, B.: On the use of model-driven test process specification and enactment by metamodelling foundation. In: Proceedings of IADIS International Conference Applied Computing, Rio de Janeiro, vol. i, pp. 51–58 (2011)

Magalhães, A.P., David, J.M.N., Maciel, R.S.P., da Silva, B.C., Silva, F.A.: Modden: an integrated approach for model driven development and software product line processes. In: V Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS 2011), Sao Paulo, pp. 21–30 (2011)

Matinnejad, R., Ramsin, R., An analytical review of process-centered software engineering environments. In: Proceedings of IEEE 19th International Conference and Workshops on Engineering of Computer-Based Systems, pp. 64–73 (2012)

Mingsong, C., Xiaokang, Q., Xuandong, L.: Automatic test case generation for UML activity diagrams. In: Proc. of the International Workshop on Automation of Software Test, pp. 2–8 (2006)

Mohagheghi, P., Dehlen, V.: Where is the proof?—a review of experiences from applying MDE in industry. In: Proceedings of the 4th European Conference on Model Driven Architecture, ECMDA-FA '08, pp. 432–443. Springer, Berlin (2008)

Montoni, M., et al.: Taba workstation: supporting software process deployment based on CMMI and MR-MPS.BR. In: Product-Focused Software Process Improvement. LNCS, pp. 249–262. Springer, Berlin (2006)

Mussa, M., Ouchani, S., Sammane, W., Hamou-Lhadj, A.: A survey of model-driven testing techniques. In: Ninth International Conference on Quality Software, pp. 167–172 (2009)

OMG: MDA Guide. Version 1.0.1 (omg/2003-06-01) (2003)

OMG: UML 2.0 Testing Profile, Final Adopted Specification. Version 1.0, July (2005). Available at: http://www.omg.org/spec/UTP/1.0/IEEE2008

OMG: Software Process Engineering Metamodel Specification, Version 2.0 (2008)

OpenUP Component—MDD (2008). Available at: http://www.eclipse.org/epf/openup_component/mdd.php

PMI: A Guide to the Project Management Body of Knowledge, 4th edn. Project Management Institute (PMI), Newtown Square (2008)

Samba, D., Lbath, R., Coulette, B.: Specification and implementation of SPEM4MDE, a metamodel for MDE software processes. In: Proceedings of the 23rd International Conference on Software Engineering Knowledge Engineering (SEKE'2011), Miami Beach, pp. 646–653 (2011)

Wang, H., Zhang, D.: MDA-based development of e-learning system. In: Proc. 27th International Computer Software and Applications Conference, Texas, California p. 684. IEEE Press, New York (2003)

Weber, S., Emrich, A., Broschart, J., Ras, E., Ünalan, Ö.: Supporting software development teams with a semantic process and artifact-oriented collaboration environment. In: Proc. SOFTEAM'09 (2009)

Yuan, Q., Wu, J., Liu, C., Zhang, Z.: A model driven approach toward business process test case generation. In: Proc. of the 10th International Symposium on Web Site Evolution (WSE), pp. 41–44 (2008)

Zamli, K.Z., Mat Isa, N.A., Khamis, N.: The design and implementation of the VRPML support environments. Malays. J. Comput. Sci. **18**(1), 57–69 (2005)