2011 18th Asia-Pacific Software Engineering Conference

Towards a Framework for the Application of Model-Driven Development in Situational Method Engineering

Zahra Zohrevand, Yusef Mehrdad Bibalan, Raman Ramsin
Department of Computer Engineering
Sharif University of Technology
Tehran, Iran
e-mail: zohrevand@ce.sharif.edu, bibalan@alum.sharif.edu, ramsin@sharif.edu

Abstract—**Model-Driven Development (MDD) is a promising approach to software development, mainly due to the pivotal role of models in its process, and the high potential it provides for automated model/software generation. Since software processes can themselves be considered as software, any technique or approach applied in the context of software development is also of potential use in the development of software development processes. Accordingly, the MDD approach can potentially be applied in the context of Situational Method Engineering (SME) − a branch devoted to the study of developing bespoke software development processes and methods based on the specific characteristics of the development project at hand.**

The authors propose a Model-Driven Situational Method Engineering (MDSME) framework by defining different levels of abstraction for process models, in compliance to the multi-level model abstraction and transformation approach prescribed by the Model-Driven Architecture (MDA). The framework can be used by method engineers to construct custom software development methodologies through an MDD approach that facilitates automation, enhances the accuracy of the development process produced, and improves the portability of process models.

Keywords–Situational Method Engineering; Model-Driven Development; Model-Driven Architecture; Model Transformation

I. INTRODUCTION

As a result of widespread use of complex software systems, Software Development Processes (SDPs) have become highly complex. This has in turn resulted in the emergence of the Method Engineering (ME) discipline, which is aimed at studying, designing, and developing custom SDPs. Situational Method Engineering (SME) is one of the most prevalent branches of ME which focuses on the development of a custom SDP based on the specific requirements of the target organization and the project situation at hand [1].

Software Engineering (SE) is continuously becoming enriched with new techniques and approaches; applying new techniques as well as comparing them with existing ones can be essential for finding suitable methods to deal with ever-increasing challenges in SE contexts. In SME, as in SE, we can eliminate existing problems through examining new approaches. Compared to SE, SME is an immature discipline, and therefore suffers from numerous unresolved problems, including accuracy, portability, and automation.

Model-Driven Development (MDD) has been recognized as an effective SE approach for enhancing portability and automation in SDPs. In MDD, models are the essential software artifacts, from which the final software code is generated [2]. Model-Driven Architecture (MDA) is the OMG's particular vision of MDD. MDA can be regarded as a subset of MDD which encompasses three levels of modeling, each producing models at a different level of abstraction: Computation-Independent Model (CIM), Platform-Independent Model (PIM), and Platform-Specific Model (PSM). In MDA, software development is performed through transforming the CIM into the PIM, the PIM into the PSM, and the PSM into code, with a particular emphasis on increasing the level of automation [3]. The information of an enterprise system is usually obtained from various sources, e.g. system engineers, managers, software engineers, and quality engineers, each representing specific views and levels of abstraction [4]. They subsequently have to be integrated into a consistent form; this can be effectively addressed through applying different levels of modeling and model transformation, as prescribed by MDA.

Due to its positive effect on portability and automation, MDD is a promising means for ameliorating the status quo in SME. In this paper, the authors investigate the possibility of using MDD in SME, and propose a Model-Driven Situational Method Engineering framework (MDSME) for applying MDD in SME projects. The proposed framework encompasses four modeling levels, specifically targeted at SME contexts: Enactment-Independent Model (EIM), Paradigm-Independent Model (ParIM), Paradigm-Specific Model (ParSM), and Platform-Specific Model (PSM). Process models can thus obtain an enhanced level of portability, since high-level models can be mapped to different platforms. Required model transformations and approaches for their automation have also been addressed in this paper, so that the burden of SME activities is passed from method engineers to tools, thereby increasing accuracy and development agility.

In addition to enhancing portability and automation, this framework promises to also resolve certain other problems that currently afflict the SME discipline: Most SME approaches are subjective, whereas in MDSME, due to the specific characteristics of MDD, all ME activities are positioned within a well-defined engineering process;

furthermore, in this framework, methods will be engineered through enriching models, thus making use of all the advantages that multilevel modeling has to offer.

Having delineated the outline of our approach, the rest of this paper is organized as follows: Section 2 briefly reviews MDD and SME concepts and the research related to this work; Section 3 proposes an MDD framework for SME (MDSME) by delineating modeling levels, and complements the framework with definitions for the model transformations required between these modeling levels; Section 4 evaluates MDSME via comparing it with MDD frameworks that are used in other contexts, as well as with existing SME approaches; and Section 5 presents the concluding remarks and discusses possible directions for furthering this research.

II. Concepts and Related Research

The set of concepts and the research related to our proposed framework (MDSME) span two distinct areas: *A*) MDD, as used in contexts other than SME, and *B*) existing SME approaches. We will focus on these areas in separate subsections.

A. Model Driven Development

In MDD, models have a pivotal role: software implementations are generated through applying transformation mechanisms on models. The widespread adoption of MDD is mainly due to its potential for automatic implementation and code generation, thus eliminating repetitive manual low-level tasks. In MDD, these tasks are encapsulated in the form of transformations. MDD can therefore reduce development costs while improving software consistency, maintainability and quality [2]. Changes in implementation strategies can be achieved through modifying the transformations. Shifting engineering concerns to platform-independent levels allows developers to focus on designing applications without involvement in platform- specific concepts.

MDD has already been applied to contexts other than software engineering: The approach in [4] investigates the applicability of MDD to process engineering by using new meta-modeling techniques; it addresses the organization of process meta-models, their relationships with other meta-models, and mechanisms for producing executable platform-specific workflows from generic business processes. Since the advent of MDD, it has been applied to various areas of software engineering: It has been shown that the model-driven approach is suitable for use in a Service-Oriented context [5], [6]; its utilization for Web Engineering has also been reported [7], [8], [9]; and its applicability has been explored in the development of Context-Aware systems [10], [11], [12].

B. Situational Method Engineering

It has been observed that different organizational settings and projects requirements cannot be satisfied by any single software development methodology. This is the driving force behind Situational Method Engineering (SME), which promotes the idea of retrieving, adapting, assembling, and tailoring process components for engineering made-to-order methodologies; methodologies that are tailored to fit the specific situations of the development project at hand [13].

Various approaches have been proposed for SME, including: Paradigm-based [14], Extension-based [15], Assembly-based [16], and a combination of Assembly-based and Roadmap-Driven[16]. None of them, however, explicitly addresses SME's accuracy, portability, and automation problems. In [17] and[18], two SME approaches are reported which propose the application of modeling in method engineering; the approach proposed in [18] even leads to a Computer-Aided Method Engineering (CAME) tool. However, they fail to address the problems of accuracy and portability, mainly due to their subjective processes.

III. MDSME Framework

We propose MDSME as a framework for applying MDD in SME. This approach is novel in two main aspects:

- **It uses MDD in a novel context:** MDD has thus far been only used in the context of software and process engineering [4]. We define MDD modeling levels and transformations that are specifically intended for SME.
- **It proposes a new approach for SME:** MDSME is specifically intended for addressing SME problems.

The MDSME framework will be explained throughout the rest of this section. However, before delving into the details, the fundamental components of the framework will be introduced; these are the components that should be included an MDD framework to guarantee its completeness. These components have been identified through studying MDD approaches which have been proposed and applied in various contexts (as referred to in section 2) and a number of other MDD frameworks that have been used in software engineering[19], [20],[21]. Based on this study, the necessary components of an MDD framework include *modeling levels,* and the necessary *transformations* between these models. Accordingly, four levels of modeling have been identified for our proposed MDSME framework. These levels, and the necessary transformations between them, are described in the following sections.

A. MDSME Framework: Modeling Levels

In order to define MDSME modeling levels, accurate boundaries have to be distinguished between them.

1) Boundaries of modeling levels

To identify boundaries for our modeling levels, we first explore the boundaries defined in the MDA approach, as practiced in software engineering; these will then be mapped into the SME context. In MDA, the boundaries between the modeling levels are based on two important concepts: *Computation* and *Platform*. In order to delineate boundaries for the levels of modeling in SME, we will first have to map these two concepts to the SME context.

a) Computation in SME

Computation is a distinguishing feature in MDD, defining the boundary between the Computation-Independent Model (CIM) and the Platform-Independent Model (PIM). The requirements of the system, as well as the situation in which the system will be used, are modeled in the CIM. Information about automated data processing systems and their implementation details are also concealed in the CIM [3]. CIM thus shows the relationships between the system and its environment as to the expectations from the system, whereas PIM is dependent on the implementation features of the system regardless of platform specifications.

The definitions of CIM and PIM reveal that the software system, which is the final product of the MDD process, plays the role of a computational element. On the other hand, in SME, the software development methodology is the final product; thus, it corresponds to the software system that is produced in SE. However, we cannot consider the produced methodology as a computational element in the MDSME framework, because it does not have a role concerned with automation in its environment (except where methods must be executable, such as in a Process-Centered Software Engineering Environment–PSEE). The engineered methodology, along with other extant control rules, will be applied as the management rules that govern the organization. Therefore, we have replaced *computation* with *enactment* – a set of principles and practical rules which can be independently used to control a software engineering process without conflict; the produced methodology plays the role of an enactable entity in the context of SME. Accordingly, CIM is replaced with EIM (Enactment-Independent Model) which models the organizational behavior and the expectations from the methodology.

b) Platform in SME

In MDD, the notion of *platform* separates the Platform-Independent Model (PIM) from the Platform-Specific Model (PSM). It may denote various types of concepts, such as: execution environments, programming languages, and constraints on firmware or hardware. There may be different levels of abstraction for platforms. That is, we might have a PSM that is independent from a certain platform [3]. In other words, based on the abstraction levels of platforms, there may exist various levels of PSM: PSM_1, PSM_2, …, PSM_n.

The same holds true for SME: There are different method platforms, even though they have differences with their SE counterparts. In this regard, method models may have to abide by various constraints on elements such as: software modeling language (such as UML), the method Process Modeling Language (such as UPM or PROMENADE), and the method development environment; selection of these elements may be based on the project situation identified.

During the process of mapping between SE platforms and their SME counterparts, situations were encountered that couldn't be mapped to a specific platform, even though they had a higher priority in comparison to other platforms. These situations denoted high-level concepts of software methodologies which specify the methodology's *paradigm*. They thus constitute a new constraint in SME that should be applied before the *platform*. We will describe this constraint in the following section.

Paradigm

Various definitions have been proposed for the notion of *paradigm*[22], [23], [24], [25], but none of them offer a comprehensive definition of the concept. In[22], descriptions have been offered for the *Engineering Paradigm* and the *Scientific Paradigm*; each of these paradigms can be used for Engineering (practical) or Scientific purposes. Methodology modeling conforms to the situation where an Engineering Paradigm is being used for an Engineering purpose: it proposes an engineering approach which will be used in an engineering context (SE). In this situation, a paradigm is used as an engineering tool and is equivalent to models or patterns that guide us through modeling in software development[22]. Method paradigms can have a spectrum of effects on the method: From coarse-grained (such as specifying the method's viewpoint to real-world entities, as in the agent-oriented approach) to fine-grained (such as prescribing that a simple *Add* operation be carried out in a structured way).

To apply a paradigm that is related to one or more situations, its general model has to be designed (if not already available), and should then be combined with the methodology's structure. Paradigm models can be presented in various forms. For example, some may be modeled as a meta-model (e.g., object-oriented paradigm), while others may be presented in the form of a set of rules and restrictions (e.g., formal aspects of a technique).

2) MDSME Modeling levels

Based on the boundaries defined for MDA modeling levels and their mappings to SME, four levels of models have been proposed for MDSME:

- **EIM (Enactment-Independent Model)**: Enterprise process model that is independent of enactment and method concerns.
- **ParIM (Paradigm-Independent Model)**: Method model independent of any particular paradigm.
- **ParSM (Paradigm-Specific Model)**: Method model based on a particular paradigm, but independent of any particular platform.
- **PSM (Platform-Specific Model)**: Method model based on a particular platform.

These modeling levels are described in more detail in the following subsections.

a) EIM

At this level, the enterprise process will be modeled as independent of enactment concerns. The models required at the EIM level include:

- **As-Is process model**: this model is produced to identify the environment workflow and the business process within which the current software development methodology is applied.
  - Different notations, such as BPMN or UML, can be used for this purpose. As an example, we have conducted a case study in which the behavioral aspects and interactions of the enterprise process have been modeled using a UML activity diagram (as shown in Fig. 1). Analogously, class diagrams can be used to model the structural aspects of the enterprise process.

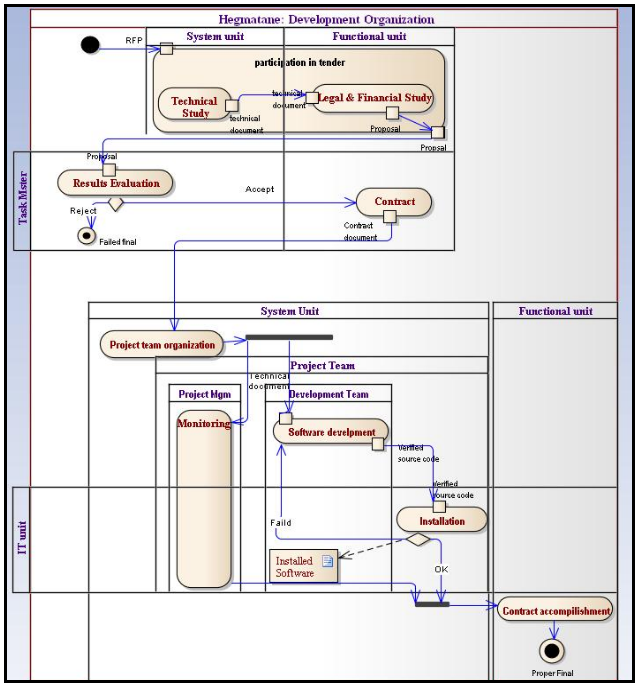

Figure 1. Example of an Enactment Independent Model

- **To-Be Process Model**: This model defines what the future enterprise process should be. The boundary between methodology and organizational processes, as well as their interactions, is determined in this model.
  - This model can be depicted by the same diagram types as those used for modeling the As-Is Process.
- **Method Situation Model**: this model portrays the desirable situations and their relationships, and spans the method's functional and non-functional situations. This model can be extracted from the *To-Be Process Model*.
  - The situation model of our case study is depicted using a new diagram (shown in Fig. 2), the metamodel of which is a subset of the class diagram's metamodel.

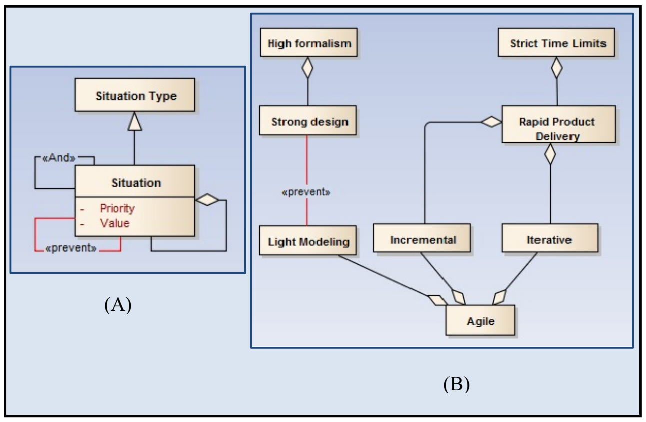

Figure 2. A) Method Situation Metamodel, B) Example of a Method Situation Model

b) ParIM and ParSM

As explained in the previous section, *Paradigm* is what distinguishes ParIM from ParSM and ParIM is a methodology model which is produced regardless of any specific paradigm concerns. It can be represented at several levels of abstraction (named as $ParIM_1$, $ParIM_2$, …, $ParIM_n$). The $Paradigm_i$ independent model, which is $ParIM_i$ and $ParSM_i$ at the same time, can be considered as the methodology architecture in relation to $Paradigm_i$, since it depicts a relatively higher-level view of the method. After applying $Paradigm_i$ to $ParIM_i$, it becomes $ParSM_i$ or $ParIM_{i+1}$, which can be considered as the detailed design model of the methodology regarding this paradigm. Thus, analogous to the levels defined for ParIM, ParSM is also constructed at different levels of abstraction ($ParSM_1$, $ParSM_2$, …, $ParSM_n$). The models required at the ParIM/ParSM level include:

- **Structural Model of the Method:** This model portrays method elements, e.g. roles and activities, and their positions relative to each other.
  - Any Process Modeling Language (PML) can be used for this purpose. In our case study, we have used the Unified Process Model (UPM) [26] as the PML; in UPM, class diagrams are used for modeling structural aspects. Fig. 3 depicts an example.
- **Behavioral Model of the Method:** This model portrays the process of method by depicting how the internal elements interact to achieve the desired objectives.
  - When using UPM, the behavioral aspects can be depicted using an activity diagram, as shown in Fig. 4.

c) PSM

At this level, the method model is bound to one or more items of the method engineering platform. Similar to ParSM, there may be several levels of PSM, named as: PSM_1, …, PSM_m. PSM_m is the final model of this type, representing the produced methodology which should be applicable in the target organization.

The types of diagrams used for representing PSM depend on the elements of the platform, and the same types of diagrams are not necessarily used at all levels. However, structural and behavioral models of the method should be provided at each level. Moreover, the PML should also have facilities for modeling the metamodels of a software modeling language at the appropriate level (where the notation should be determined for the target methodology).

B. MDSME Framework: Transformations

Model transformation spans both model-to-model and model-to-code transformations. This section presents the different types of transformations required between the MDSME framework's modeling levels, as well as the appropriate approaches for executing them. To this aim, a set of transformation features will first be specified. The types of these transformations will then be expressed based on the specific features of the modeling levels in MDSME, and the transformation approaches will be analyzed and selected.

1) *MDSME Model Transformation Features*

Model transformation approaches are classified in various categories, based on their specific features. Sets of these features have been presented in [27], [28] and [29]. Although these are provided in a software development context, they have been defined based on the engineering aspects of process models. On the other hand, they are independent of the products of the engineering process. Therefore, comparison and selection of model transformation approaches in SME can be performed with regard to the same features. Proposed approaches in this framework, which have been selected based on the features presented in [29], include: **Specification**, **Transformation Rules**, **Rule Application Control**, **Rule Organization**, **Source-Target Relationship**, **Directionality**, **Tracing, and Incrementality**.

In SME, there are certain important issues that should be addressed in transformations:

- The process aspect of the methodology is very significant, and the behavioral interactions occurring at different levels of granularity can be quite complicated. Thus, in SME, transformation approaches should be able to cope with complex interactions.
- Practicality of a methodology and its deficiencies are usually detected during the application of the methodology. Thus, providing facilities for improving the methodology is a necessary feature for SME approaches. To this aim, *Incrementality* is very important in this context, so that the method can be improved just by modifying the Situation Model.

2) *Model Transformations*

This section presents the transformations required in the MDSME framework. The most appropriate transformation approaches are then proposed based on the classification presented in [29]. These transformation approaches are classified into two main categories, which are consistent with the classification provided by [28].

a) Vertical transformation

This category includes inter-level transformations where the source and target models are at different levels of abstraction; namely:

- **EIM to ParIM**: ParIM resides at a lower abstraction level than EIM. This type of transformation often requires changes to the method PML and therefore it can be considered as an *Exogenous* transformation [28]. The transformation of the models depicted in Fig. 1 and Fig. 2 into the model of Fig. 3 is of this type.
  - Transformation from EIM to ParIM requires decision-making solutions which are often associated with the change of language. To automate solution-based transformations in this context, the *pattern-based* approach is proposed. It can be combined with the *structure-based* approach, which provides facilities for accommodating the change of language.
- **ParIM to ParSM**: This type of transformation will be carried out based on a paradigm; therefore, a paradigm model should be applied to the methodology. It can therefore be categorized as a *Model Merging Transformation* [3], and since it is a *Vertical* transformation in the same process modeling language, it can be considered as *Formal Refinement* [28].
  - Based on the paradigm model, the appropriate transformation approach may differ. For example, if the metamodel of the elements after applying the paradigm exists, then the *structure-based* approach would be useful, since this approach is based on the hierarchical structure of the source and target models, and these metamodels provide mapping facilities for applying the transformation. When the paradigm model is represented in the form of a set of constraints and relationships, a *declarative* approach (such as *Relational*) is more appropriate, since it is not necessary to address how to perform the conversions. Rather, it only needs to present the target model in the form of a set of rules and relationships, which can be obtained from the paradigm model. Fig. 5 and Fig. 6 show ParSM diagrams which are the results of transforming the ParIM models shown in Fig. 3 and Fig. 4, respectively; *Delivery Strategy* is the paradigm applied.

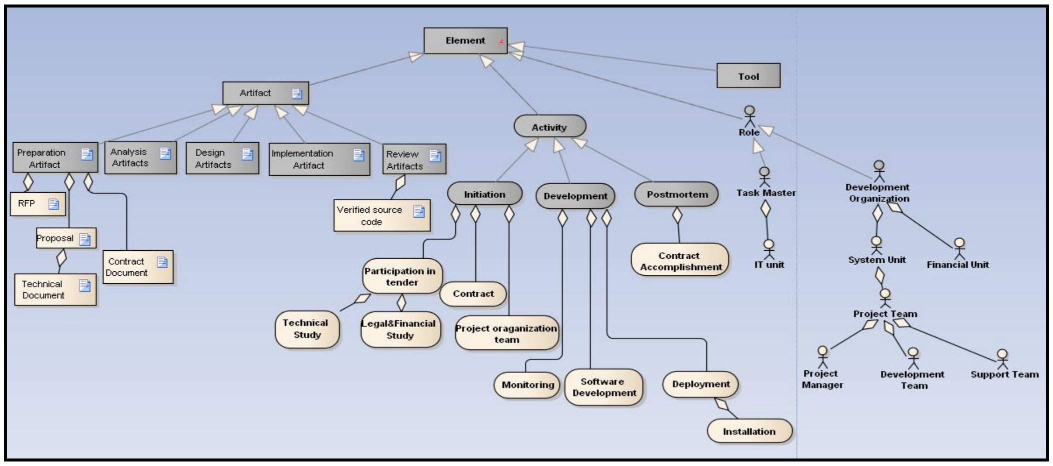

Figure 3. Example of a Structural Model

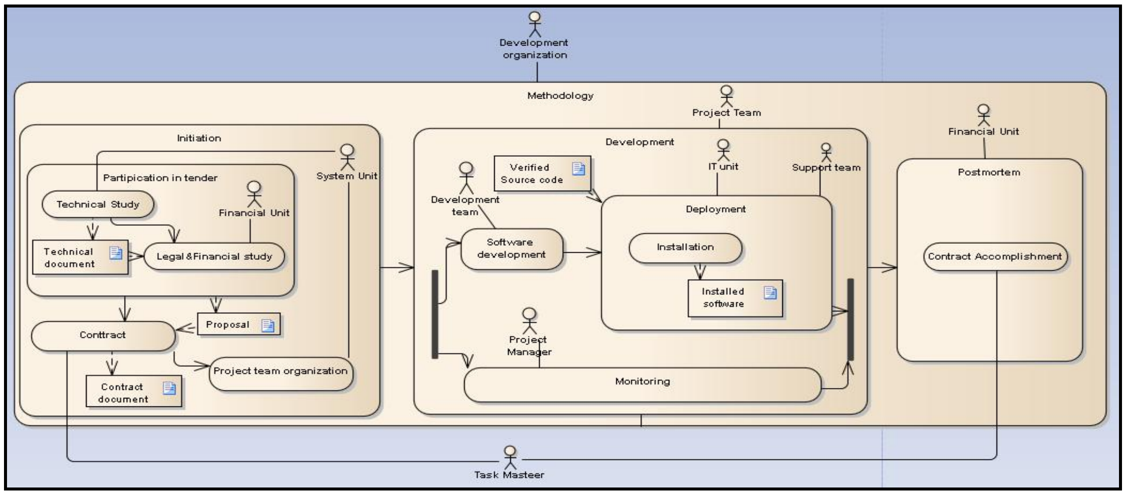

Figure 4. Example of a Behavioral Model

- **ParSM to PSM** (with a change in abstraction level): This type of transformation is also *Exogenous*, so it can be considered as *Code Generation* based on [28].
  - o The suitable approach for this type of transformation can be quite different depending on the type of platform. For example, transition from a method PML to another can be easily done via structure-based approaches. Whereas the situation is quite different for the definition of a software modeling language: if the goal is the creation of a new modeling language, then there will be a very low potential for automation; thus, a facility to define and carry out transformations using *Direct Manipulation* or the *Operational* approach is recommended. On the other hand, where the use of existing modeling languages is being considered, the *pattern-based* approach will be advisable.

b) Horizontal transformation

This category includes all types of intra-level transformations.

- **Model Refinement**: This type of transformation is applied to restructure the models in order to improve them (also called *Refactoring*) [28].
  - o Since model refinement is mainly based on human decisions and comparisons, provisions are required for facilitating selection and replacement. *Direct Manipulation* or the *Operational* approach can be utilized to create the appropriate interfaces.
- **Diagram conversion**: As a result of this type of transformation, diagrams will be created at the same level of abstraction. It is used to simplify other types of transformations, or to display other aspects of existing models.
  - o Since all diagrams should be compatible with each other, multidirectional transformation methods, such as Relational approaches (which use multi-directional rules with high incrementality), are more desirable.

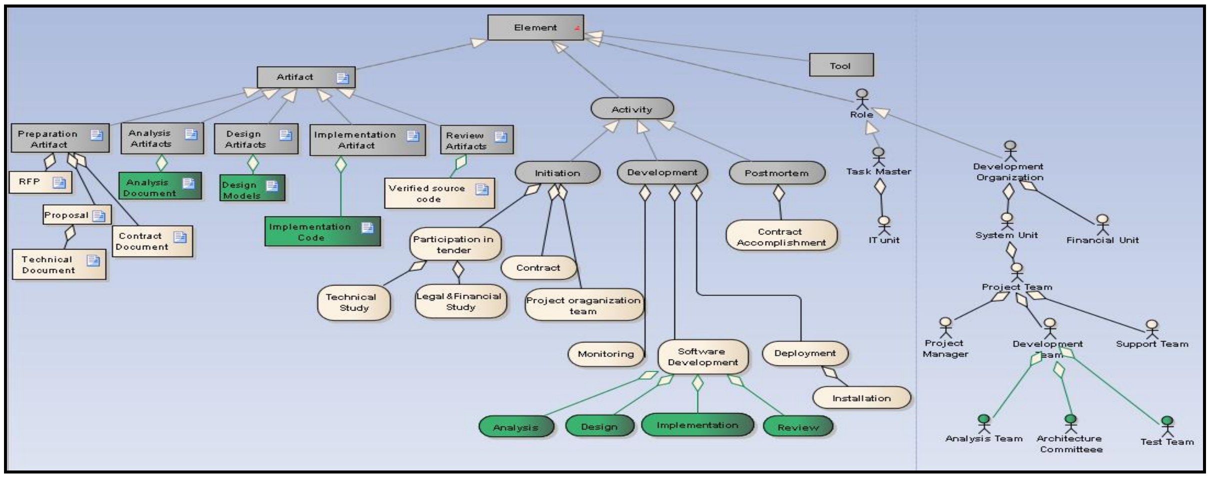

Figure 5. Example of a ParSM Structural Model: Result of transforming the ParIM model shown in Fig. 3

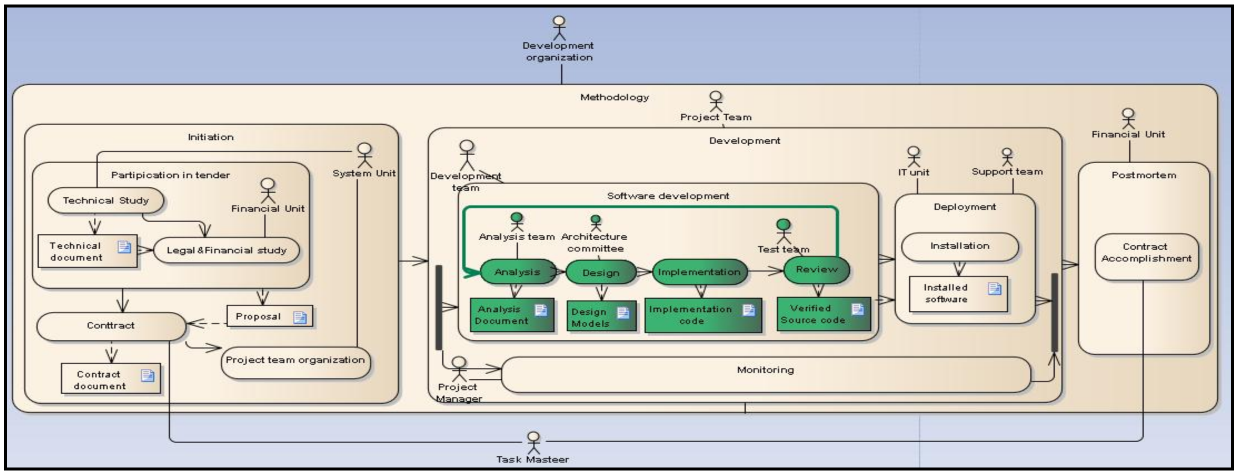

Figure 6. Example of a ParSM Behavioral Model : Result of transforming the ParIM model shown in Fig. 4

IV. MDSME Framework Evaluation

The MDSME framework can be considered both as a new MDD framework and also as a new approach to SME. In this section, MDSME is evaluated through 1) comparison with other MDD frameworks, and 2) comparison with other SME approaches.

A. Comparison of MDSME with other MDD frameworks

Unfortunately, an all-inclusive set of criteria for evaluating MDD frameworks is not available. This comparison is therefore carried out based on a set of basic principles and rules which any MDD approach should adhere to. The results of this evaluation are shown in table I.

B. Comparison of MDSME with other SME approaches

There is no a standard set of criteria for comparing and evaluating different SME approaches. In[30], a set of evaluation criteria has been proposed for comparing two SME approaches, but these criteria are specifically aimed at *assembly-based* approaches. Thus, in this section, the process evaluation criteria defined for SE have been mapped to the SME context. The results of the evaluation are shown in table II. Moreover, scrutinizing the patterns of SME [31] (presented in the form of a framework), reveals that there is a design seam in fragment-based SME between *Method Initiation* and *Method Construction*; this shortcoming can be adequately addressed by MDSME's ParIM/ParSM models.

Table I. MDSME Framework in comparison to other MDD frameworks

	MDA	***Model Driven Process Engineering [4]***	***MDSME***
Approach to definition of levels	Viewpoint abstraction: • Business viewpoint • System viewpoint • Software viewpoint	Linguistic metamodeling	Viewpoint abstraction: • Enactment-independent viewpoint • Paradigm-independent viewpoint • Paradigm-specific viewpoint • Platform-specific viewpoint
Transformation Type	• Vertical • Horizontal	• Horizontal (If abstraction level of target language is lower, then can be Vertical)	• Vertical • Horizontal
Problem-to-solution transformation automation potential	Low	N/A	High
Portability to	• Platforms (in solution domain)	• Process execution environment (particularly PML)	• Method platforms (in solution domain) • Situations (in the transition to solution domain for each situation)

Table II. MDSME Framework in comparison to other SME approaches

	Generic process for SME [1]	***MDSME Framework***	***MEMA-Model [17]***	***Eng'ng Method from MRS [18]***
Design Model	N/A (Seamed)	ParIM$_1$,…, ParIM$_n$ (multilevel design)	Semi-open method	Decisional metamodel
Potential for Process Automation	Low (selection and use of method fragments)	High (systematic transformations)	N/A	High
Portability of Method model	Low (limited to paradigm-based approach)	High (hierarchy of situations and method platforms)	Medium to High (project situations)	Medium (relation types and detailed descriptions)
Complexity management	Weak	Medium	Medium	Medium
Maintainability	Low (due to its requirements-to-components mapping approach)	High (automation capabilities, strong design)	Medium to High	High (metamodeling approach)
Environment and tool dependency	Low (due to need for repository)	High dependency (transformation tools)	Low (manual process)	High dependency (instantiation environment)

V. Conclusions and Future Work

Development methodologies such as MASTER, C3, and MODA have employed the principles and concepts of Model-Driven Architecture in order to take advantage of the approach in software engineering. In this paper, a new framework for Model-Driven SME (MDSME) is proposed which provides the means to engineer suitable methods for specific situations by employing the MDD approach.

Similar to other MDD approaches, MDSME models are not just used as documentation for engineering. Rather, they have a fundamental role in development processes through which the final product will be obtained by following specific steps. The shift from implicit modeling to explicit modeling is a characteristic of every model-based approach [4]. This, along with the different modeling levels of the proposed framework, helps SME activities to be expressed more clearly. Also, by implementing the identified transformations, SME activities will be automated. Increased accuracy and production speed for SME are therefore other advantages of this framework. Portability of the method models is also increased through multilevel modeling.

To utilize the MDSME framework, concrete processes have to be composed based on it. Future research can focus on proposing these MDSME-based processes; automated tools can then be developed for applying the framework.

Acknowledgement

We wish to thank Mr. Massood Khaari for assisting in the revision of this paper.

References

[1] B. Henderson-Sellers and J. Ralyte, "Situational Method Engineering: State-of-the-Art Review", J. Universal Computer Science, 16(3), 2010, pp. 424-478.

[2] P. Swithinbank et al., "Patterns: Model-Driven Development Using IBM Rational Software Architect", IBM Redbook, 2005.

[3] J. Miller and J. Mukerji, "MDA Guide: Version 1.0.1", 2003, Available online at: http://www.omg.org/cgi-bin/apps/doc?omg/03-06-01.pdf [Accessed: May 2011].

[4] E. Breton and J. B´ezivin, "Model driven process engineering", Proc. 25th IEEE Annual International Computer Software and Applications Conference (COMPSAC'01), 2001, pp. 225–230.

[5] C. Emig, K. Krutz, S. Link, C. Momm, and S. Abeck., "Model-driven development of SOA services", Technical Report, Forschungsbericht, 2007.

[6] P. Mayer, A. Schroeder, and N. Koch, "MDD4SOA: Model-Driven Service Orchestration", Proc. 12th IEEE International Enterprise Distributed Object Computing Conference (EDOC'08), 2008, pp. 203-212.

[7] X. Qafmolla, "Automation of Web Services Development Using Model Driven Techniques", Proc. ICCAE'10, 2010, pp. 190-194.

[8] F. Daniel, "Context-Aware Applications for the Web: A Model-Driven Development Approach", Context-Aware Mobile and Ubiquitous Computing for Enhanced Usability–Adaptive Technologies and Applications: IGI Global, 2009, pp. 59-82.

[9] J. Escalona and A. Gustavo, "NDT: A Model-Driven Approach for Web Requirements", J. IEEE Transactions on Software Engineering, 34(3), 2008, pp. 377-390.

[10] D. Ayed, D. Delanote, and Y. Berbers, "MDD Approach for the Development of Context-Aware Applications", Proc. 6th International and Interdisciplinary Conference on Modeling and Using Context (CONTEXT'07), LNAI 4635, 2007, pp. 15–28.

[11] Z. Jaroucheh, X. Liu, and S. Smith, "Apto: A MDD-Based Generic Framework for Context-Aware Deeply Adaptive Service-Based Processes", Proc. 8th IEEE International Conference on Web Services (ICWS'10), 2010, pp. 219-226.

[12] E. Serral, P. Valderas, and V. Pelechano, "Towards Model Driven Development of Context-Aware Pervasive Systems", J. Pervasive and Mobile Computing, 6(2), 2010, pp. 254-280.

[13] J. Ralyté, S. Brinkkemper, B. Henderson-Sellers, "Situational Method Engineering: Fundamentals and Experiences", IFIP-Series 244, Springer, 2007.

[14] J. Ralyté, R. Deneckère, and C. Rolland, "Towards a Generic Method for Situational Method Engineering", Proc. 15th International Conference on Advanced Information Systems Engineering (CAiSE'03), LNCS 2681, 2003, pp. 95-110.

[15] R. Deneckere, "Approche d'extension de méthodes fondée sur l'utilisation de composants génériques", PhD Thesis, University of Paris 1-Sorbonne, 2001.

[16] I. Mirbel and J. Ralyté, "Situational Method Engineering: Combining Assembly-Based and Roadmap-Driven Approaches", J. Requirements Engineering, 11(1) , 2006, pp. 58–78.

[17] T. Punter and K. Lemmen, "The MEMA-Model: Towards a New Approach for Method Engineering", J. Information and Software Technology, 38(4), 1996, pp. 295-305.

[18] D. Gupta and N. Prakash, "Engineering Methods from Method Requirements Specifications", J. Requirements Engineering, 6(3), 2001, pp. 135-160.

[19] M. Asadi and R. Ramsin, "MDA-Based Methodologies: An Analytical Survey", Proc. 4th European Conference on MDA Foundations and Applications (ECMDA-FA'08), LNCS 5095, 2008, pp. 419-431.

[20] J. B´ezivin and O. Gerb´e, "Towards a Precise Definition of the OMG/MDA Framework", Proc. 16th International Conference on Automated Software Engineering (ASE'01), 2001, pp. 273–280.

[21] L. Bastida, and et al., "Model-Driven Methodology and Architecture Specification", SHAPE Project, 2006, Available online at: http://www.shape-project.eu/work-packages [Accessed: April 2010].

[22] C. Cares, X. Franch, and E. Mayol, "Perspectives about Paradigms in Software Engineering", Proc. 2nd Workshop on Philosophical Foundations of Information Systems Engineering (PHISE'06), 2006, pp. 737-744.

[23] E. Göktürk and N. Akkok, "Paradigm and Software Engineering", Proc. Workshop on Impact of Software Process on Quality (IMPROQ'04), 2004, pp. 10-17.

[24] S.H. Kaisler "Software Paradigms", John Wiley, 2005.

[25] T.S. Kuhn, "The Structure of Scientific Revolutions", 3rd ed., University of Chicago Press, 1996.

[26] Object Management Group (OMG), "The Unified Process Model (UPM)", 2000.

[27] K. Czarnecki and S. Helsen, "Classification of Model Transformation Approach*es*", Proc. OOPSLA'03: Workshop on Generative Techniques in the Context of MDA, 2003.

[28] T. Mens, K. Czarnecki, and P. van Gorp., "A Taxonomy of Model Transformation*s*", Proc. Dagstuhl Seminar: Language Engineering for Model-Driven Software Development, 2005.

[29] K. Czarnecki and S. Helsen, "Feature-Based Survey of Model Transformation Approach*es*", J. IBM Systems, 45(3), 2006, pp. 621–645.

[30] N. Prakash and S.B. Goyal, "Method Architecture for Situational Method Engineering", Proc. 2nd IEEE International Conference on Research Challenges in Information Science (RCIS'08), 2008, pp. 325-336.

[31] M. Asadi and R. Ramsin, "Patterns of Situational Method Engineering", Proc. SERA'09, SCI 253, 2009, pp. 277-291.