Information and Software Technology 78 (2016) 95–120

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

A pattern-based model-driven approach for situational method engineering

Halimeh Agh, Raman Ramsin*

Department of Computer Engineering, Sharif University of Technology, Tehran, Iran

ARTICLE INFO

Article history:
Received 20 December 2015
Revised 11 May 2016
Accepted 31 May 2016
Available online 2 June 2016

Keywords:
Software development methodology
Situational method engineering
Model-driven development
Process modeling
Pattern-based model transformation

ABSTRACT

Context: Constructing bespoke software development methodologies for specific project situations has become a crucial need, giving rise to Situational Method Engineering (SME). Compared with Software Engineering, SME has a long way to go yet; SME approaches are especially deficient as to support for modeling, portability, and automation. Model-Driven Development (MDD) has been effectively used for addressing these issues in Software Engineering, and is also considered a promising approach for resolving them in SME.

Objective: This paper aims to address the shortcomings of existing SME approaches by introducing a novel MDD approach, specifically intended for SME purposes, that uses a pattern-based approach for model transformation.

Method: Developing a MDD approach for SME requires that a modeling framework, consisting of modeling levels, be defined for modeling software development methodologies. Transformation patterns should also be specified for converting the models from one level to the next. A process should then be defined for applying the framework and transformations patterns to real SME projects. The resulting MDD approach requires proper evaluation to demonstrate its applicability.

Results: A framework and a semi-automated process have been proposed that adapt pattern-based model transformation techniques for application to the methodology models used in SME. The transformation patterns have been implemented in the Medini-QVT model transformation tool, along with two supplementary method bases: one for mapping the situational factors of SME projects to requirements, and the other for mapping the requirements to method fragments. The method engineer can produce the methodology models by using the method bases and executing the transformation patterns via the tool.

Conclusion: The validity of the proposed approach has been assessed based on special evaluation criteria, and also through application to a real-world project. Evaluation results indicate that the proposed approach addresses the deficiencies of existing approaches, and satisfies the practicality requirements of SME approaches.

1. Introduction

The challenges of modern software development have necessitated that software development methodologies be custom-built or adapted; this has led to the emergence of a discipline called Method Engineering (ME), formally defined as: "The engineering discipline to design, construct and adapt methods, techniques and tools for the development of systems"; Situational Method Engineering (SME) is an important subfield of method engineering that focuses on composing or adapting a software development methodology according to the specific characteristics of the project at hand (expressed in terms of situational factors) [1]. Various high-level approaches have been proposed for SME, the most prominent of which are: Paradigm-based [2], Assembly-based [3], and Extension-based [4]. The paradigm-based approach is rooted in metamodeling, in that the target methodology is produced by instantiating, abstracting or adapting an existing metamodel; in the assembly-based approach, the target methodology is constructed by reusing methodology parts to compose a new methodology or to enhance an existing one; and in the extension-based approach, an existing methodology is augmented by applying special-purpose extension patterns. SME has been evolving for two decades; however, existing SME approaches have not been able to fully address the various requirements of the SME domain, the most important of which include: Complexity management, automation, portability, and accuracy [5]. In Model-Driven Development (MDD),

* Corresponding author. Department of Computer Engineering, Sharif University of Technology, Azadi Ave., Tehran, Iran. Fax: +98 21 6601 9246.
E-mail addresses: agh@ce.sharif.edu (H. Agh), ramsin@sharif.edu (R. Ramsin).

http://dx.doi.org/10.1016/j.infsof.2016.05.010
0950-5849/

models are the pivotal concepts and artifacts, and automating the production of models through model transformations is the primary objective [6]. This mindset has many advantages, such as increasing productivity, managing complexity, and enhancing portability. As the major manifestation of MDD, the Model-Driven Architecture (MDA) prescribes the production of three models of the target software system: Computation-Independent Model (CIM), Platform-Independent Model (PIM), and Platform-Specific Model (PSM) [7]; this multi-level modeling framework, which relies on the notions of "Computation" and "Platform", has become a de-facto standard for MDD approaches. Model transformation is an important issue in MDD, as lower-level models are produced by transforming their higher-level counterparts. There are several approaches for model transformation in MDD, including graph-based, metamodel-based, by-example, and pattern-based [8].

Due to its potential benefits, MDD has already been used for SME purposes [5]; however, as discussed in Section 5.2, existing approaches are deficient in that many of them lack adequate support for automation of model transformations, and all of them are over-dependent on the method engineer's expertise for determining the method fragments suitable for constituting the target methodology. Moreover, these methods need to be further improved as to the quality attributes typically targeted in MDD: complexity management, portability, productivity, and reusability. These attributes, as applicable in the context of SME, are briefly explained below:

- Complexity management: many SME tasks are inherently complex; prominent examples include: identifying the requirements of the target methodology, specifying method fragments for each requirement, determining the relationships among the method fragments, and modeling the constructed methodology. SME approaches should provide adequate means for managing the complexity of SME tasks and the artifacts produced.
- Portability: capability to transition the produced methodology onto different platforms (enactment environments).
- Productivity: SME is typically a prelude to the main software development activities; therefore, it is important that high-quality situational methodologies be developed and maintained rapidly and cost-effectively.
- Reusability: capability to reuse software process knowledge; this can be realized through eliciting method fragments from existing methodologies and storing them in method bases which act as knowledge repositories.

We propose a novel pattern-based MDD approach for SME with the specific aim of addressing the above issues. The approach consists of: 1) a multi-level, pattern-based MDD framework for methodology modeling and transformation (which we have chosen to call PBMDD4SME, short for "Pattern-Based Model-Driven Development for Situational Method Engineering"), and 2) a process for applying the framework to the engineering of software development methodologies based on the specifications of the project situation at hand. The framework defines three levels of models and a set of pattern-based transformation rules. The transformation rules have been implemented in the Medini-QVT model transformation tool [9], and have been augmented with two supplementary method bases; one for mapping the situational factors (which express the project situation) to methodology requirements and the other for mapping the requirements to method fragments. This enables the method engineer to produce the models and compose the target methodology in a semi-automated manner through following a model-driven, tool-supported process.

We have used the pattern-based transformation approach [8] in our proposed SME framework, as it facilitates automation and can be suitably adapted to the SME context. A *pattern* is a successful *solution* to a recurring *problem* in a given *context*. In the pattern-based model transformation approach, each transformation pattern defines a general problem of interest in the source model and its corresponding solution (transformed counterpart) in the target model [10]. Several such pattern sets already exist, but they are used for model transformation in a software development context, not for SME (examples are provided in Section 2). Our proposed transformation patterns are specifically intended for use in the SME context, and have been described by using the general template suggested in [10]; the template consists of: 1) the pattern's name, purpose, motivation, and applicability, 2) the problem sets resolved by the pattern, 3) features of the solution, and 4) an illustration of how the pattern is typically applied.

In order to evaluate our proposed approach, we have compared it with other MDD frameworks and other model-driven SME approaches based on specially defined criteria. In addition, PBMDD4SME and its process have been applied to a real-world SME project, and the results have demonstrated its merits in a practical context. Evaluation results show that the proposed approach addresses the deficiencies encountered in existing SME approaches, improves on the quality attributes, and is applicable to real-world project situations.

The rest of this paper is structured as follows: Section 2 provides a survey of the related research; Section 3 introduces the proposed PBMDD4SME framework and presents the two method bases that are used in the proposed approach; Section 4 presents a process for applying the proposed framework to SME problems; Section 5 provides the results of evaluating the proposed approach; and Section 6 presents the concluding remarks and suggests ways for furthering this research.

2. Related research

Many of the concrete methods and frameworks currently used in SME, such as the OPEN methodological approach [11,12], the ISO/IEC-24,744 framework [13,14], and the MMC approach [15], are not model-driven; whereas MDD can be of significant benefit to these approaches, mainly due to its positive effect on complexity management and automation. The literature related to this research includes works on the application of MDD for SME purposes, as well as MDD works focusing on pattern-based transformation.

In general, there are two categories of SME works related to this research:

1. SME approaches that use MDD explicitly: In [16], a methodological MDD framework is proposed for SME that is applied in three phases: Method design, method configuration, and method implementation. In [5], a MDD framework by the name of Model Driven Situational Method Engineering (MDSME) is proposed, which defines four modeling levels: Enactment-Independent Model (EIM), Paradigm-Independent Model (ParIM), Paradigm-Specific Model (ParSM), and Platform-Specific Model (PSM). In [17], methodology engineering methods based on MDD have been surveyed. In [18], MDD has been applied to method tailoring for specific project situations.
2. SME approaches that are not model-driven by name, but highly resemble classical MDD approaches: In [19], a three-level scheme has been used for describing the target methodology, which resembles the multi-level modeling scheme typically used in MDD approaches; the levels include: General level, model level, and project level. In [20], a method has been introduced to develop a methodology based on specific requirements; the method is applied in three steps, which involve multi-level specifications: Method requirements engineering, method design, and method implementation.

Several approaches for pattern-based transformation have been introduced in MDD literature, but they are typically used for model transformation in software development (not methodology development, as in SME). In [10], five essential problem-solution patterns are proposed: Mapping pattern, Refinement pattern, Node Abstraction pattern, Duality pattern, and Flattening pattern; these patterns are documented according to a description template which consists of name, goal, motivation, specification, example, and applicability. In [21], the Promotion Transformation and Instantiation Transformation patterns are proposed for multi-stage model-driven software development. In [22], the use of design patterns for model driven engineering is discussed, and two patterns (Transformation Parameters and Multiple Matching) are described as examples; the template used for describing these patterns consists of motivation, solution, and consequences. In [23], the knowledge required for transforming models from one abstraction level to the next is captured as parameterized patterns; these patterns can be used for executing the transformations in software tools. The transformation patterns proposed in [24–26] are aimed at lower abstraction levels, such as from a specific PIM to a PSM.

The main motivation for this research is the observation that existing model-driven SME approaches need to be improved in several aspects: Some of the aforementioned approaches lack adequate support for automatic model transformations (e.g., [5]), and even though complexity management is essential for the applicability of these approaches, most of them provide low-to-moderate support for this feature (e.g., [17,19,20]). Furthermore, none of the existing approaches provides a method base of reusable method parts for assembling the target methodology. In the approach that we propose in this paper, two tool-supported method bases have been developed which enable the method engineer to semi-automatically produce the methodology requirements and their corresponding method fragments through the execution of transformation patterns; reliance of the approach on the method engineer's expertise is thus significantly reduced. Even non-model-driven SME approaches and frameworks that do provide a method base (such as OPEN [11,12]) do not provide this feature.

3. Proposed framework for pattern-based MDD for SME (PBMDD4SME)

To propose a methodology engineering (SME) process based on the MDD approach, it is first necessary to define a multi-level framework for modeling in the SME context. The modeling levels defined in our proposed framework are distinguished according to the two concepts of *what* and *how*: The *what* focuses on the constituent elements of the target methodology, including its processes, roles, and work products; whereas the *how* deals with the techniques used for performing the processes of the methodology. According to these concepts, a methodology model would be gradually refined from a specification of *what* at the higher levels of modeling (abstract) to a specification of *how* at the lowest level (concrete).

Based on the general scheme outlined above, three modeling levels have been defined in our proposed framework (PBMDD4SME):

- First level: Methodology-Fragments-Independent Model (MFIM) - Models the methodology requirements and the general structure of the target methodology, consisting of high-level phases and activities.
- Second level: Technique-Independent Model (TIM) - Describes the constituents of the methodology in detail, but stops short of determining specific techniques for performing the tasks of the methodology.
- Third level: Technique-Specific Model (TSM) - Determines specific techniques for performing the tasks of the methodology.

The model at the top level is independent from the *what*, whereas the second-level model is dependent on the *what* but independent from the *how;* dependency on the *how* is introduced in the model at the third level.

As previously mentioned, the proposed framework also includes two mapping method bases, one for mapping Situational Factors to Requirements (SF2R) and the other for mapping Requirements to Method Fragments (R2MF). The modeling levels of the framework, and their constituent sub-models, are explained in Sections 3.1, 3.2, and 3.3; the method bases are explained in Sections 3.4 and 3.5. Details of how to use these models for creating a situational methodology are provided in Section 4.

3.1. MFIM level

At this level, the methodology model is independent of the fine-grained constituents of the methodology. Two models are produced at this level: one for specifying the requirements that the target methodology should satisfy, and the other for depicting the high-level phases and activities of the target methodology. The high-level constituents of the methodology are specified to address the "Scope the Methodology" requirement; this requirement is presupposed by default as a top-priority requirement for all SME projects, and should be satisfied at the MFIM level. Producing the MFIM requires more effort than the other two levels, mainly because extracting the requirements of the target methodology and resolving their mutual conflicts is carried out at this level, and also because the amount of work that is performed manually is more than that of the other levels. The models defined at this level, and the conflict resolution algorithm applied, are explained in the following sections.

3.1.1. Requirements model

This model depicts the requirements that should be met in the target methodology. Based on previous studies conducted on requirements engineering in MDD, a metamodel has been defined for modeling the requirements in the SME context. We have chosen to call this metamodel the "Requirements Engineering MetaModel for Situational Method Engineering (REMM4SME)". This metamodel, shown in Fig. 1, has been adapted from the REMM metamodel proposed in [27]. REMM has been selected for this purpose as it builds on prominent requirements metamodels such as that of SysML; however, unlike SysML, it is not dependent on any specific domain and is therefore free from the constraints that result from such dependencies.

Situational factors are the coarse-grained requirements of the target organization and the project situation at hand. These factors should be refined into finer-grained requirements; in our proposed approach, for each situational factor, a set of corresponding requirements is extracted from the SF2R method base. There is a range of possible values for each situational factor; stakeholders assign values to a factor by selecting one value from the factor's corresponding range of possible values and assigning it to the 'Value' attribute of the factor. In Fig. 1, the elements shown in italic or dashed lines have been added to the REMM metamodel in order to support the SME domain.

As seen in Fig. 1, situational factors are of three types: Environmental factors, Organizational factors, and Project factors; this particular classification of situational factors has been adapted from [28–31]. Environmental factors highlight the attributes related to the environment where the system will be operated. "Importance of project in environment", with a value range of "Yes/No", is an example of this type of situational factor; if the value "Yes" is assigned to this situational factor, requirements such as "use past

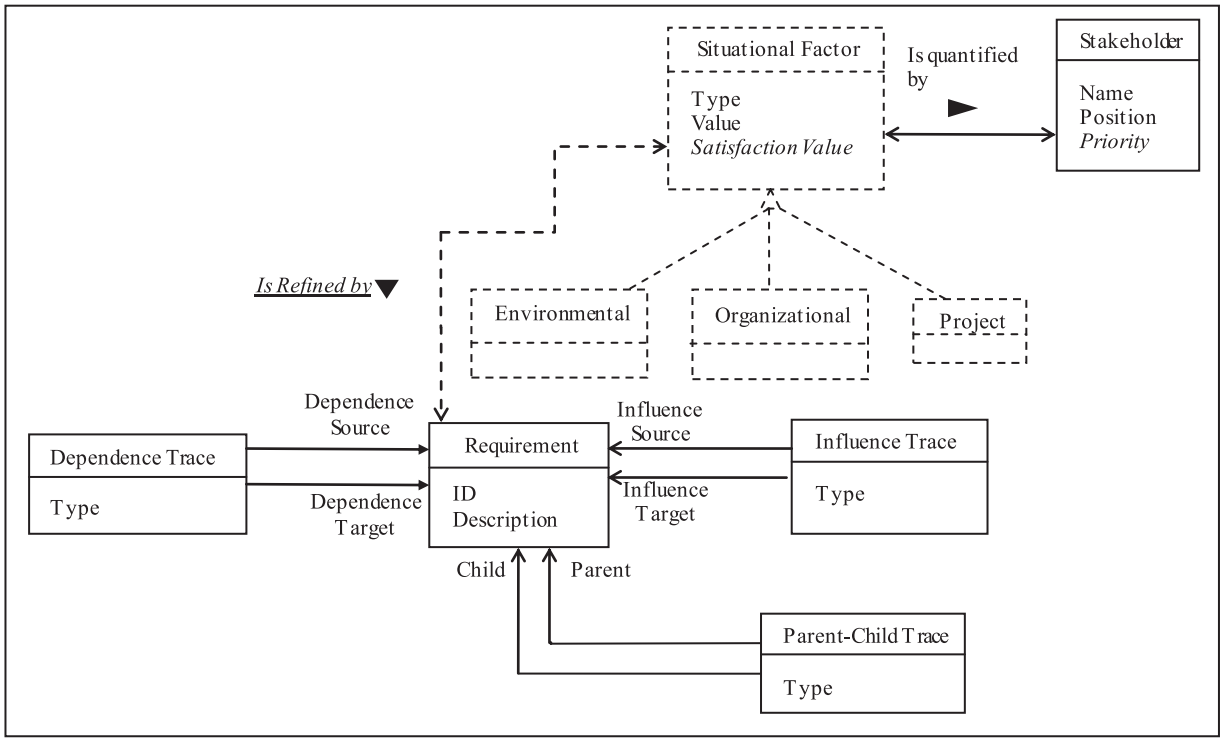

Fig. 1. REMM4SME metamodel.

project experiences" and "apply evolution (maintenance)" are extracted from the SF2R method base as refinements for the situational factor. Organizational factors determine the attributes of the organization responsible for developing the system. "Developers' business knowledge", with a value range of "Sufficient/Insufficient", is an example of this type of situational factor; if the value "Insufficient" is assigned to this situational factor, requirements such as "Perform domain analysis in each iteration" and "Improve business knowledge of team members" are extracted as its refinements. Project factors specify the attributes of the system under development. "System's dependence on UI", with a value range of "High/Low", is an example of this type of situational factor; if the value "High" is assigned to this situational factor, "Welcome change", "Apply UI-based software development", and "Enhance usability" are examples of the requirements that are extracted as its refinements.

After the set of requirements corresponding to each situational factor is extracted from the SF2R method base, a subset of these requirements is selected for satisfaction. It is possible to select the whole set of requirements; however, doing so may well result in an overly complex methodology. A Satisfaction Value (SV), which is a numerical value between 0 and 1, is calculated for each situational factor, which shows the degree to which the factor is satisfied by its corresponding requirements. The requirements selection algorithm should ensure that the selected subset maximizes the SV of the situational factor. Fuzzy Control [32] is one possible means for this purpose. This approach has long been applied in industry and has been significantly successful [33]. The contributions reported in [34] and [35] are examples of using the approach for requirements engineering: in [34], Fuzzy Logic is used for requirements prioritization; in [35], Fuzzy Inference is used for selecting the features corresponding to a specific situation.

The method proposed in [35] can be adapted and used in our approach for selecting a subset of the requirements, although any other working method may be used as well (the method engineers might even decide to do the pruning without using any specific technique, simply relying on their own expertise and experience). In [35], a feature-based RE process factory is proposed to develop RE processes based on the characteristics of the project at hand. In the Feature Analysis phase of this approach, after eliciting the values of situational factors, a Fuzzy Inference System (FIS) is used for selecting the features corresponding to a specific situation; the FIS uses a Fuzzy Rule Engine (FRE) for mapping an input space to an output space. In this approach, one or more quantified situational factors are fed to the FIS. The FRE derives a conclusion that indicates the fitness of the corresponding features according to a rule base. The logic rules in the rule base are in the form of IF-THEN statements. In our adaptation of this method, a quantified situational factor and its intended SV (0.9, for instance) are fed to the FIS as input. The FRE is configured to compute an output value for each relevant requirement, showing the degree to which it satisfies the situational factor. As an example, two of the rules for the "Schedule constraints on the project" situational factor are shown in Fig. 2. The output value can thus determine whether a corresponding requirement should be kept or removed from the final selection; for example, we may decide to keep a requirement only if its corresponding output value is greater than 0.9.

Each of the requirements is characterized by two attributes: a unique identifier (ID) and a textual description (Description). Information on the stakeholders is also recorded (Name, Position, Priority), later to be used by the Conflict Resolution Algorithm (explained in the next section).

As shown in Fig. 1, the following relationships can be defined among the requirements:

- Given two requirements R1 and R2, the following dependencies can be defined between them:
  - "R1 Requires R2". This relationship implies that R2 is needed to fulfill R1. In other words, R2 is a precondition for R1.
  - "R1 Overlaps R2". This relationship implies that the satisfaction of one of the requirements is enough and the other requirement will not be further processed.

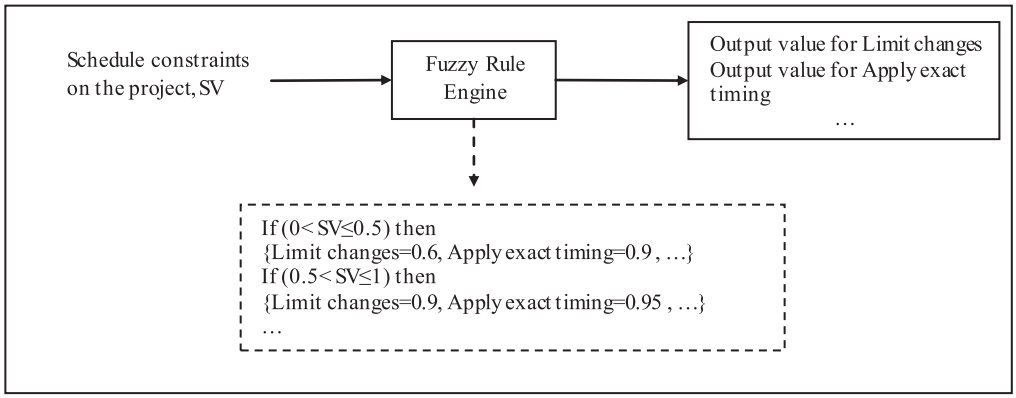

Fig. 2. Conceptual model for FIS of a situational factor (Adapted from [35]).

- Given two requirements R1 and R2, *"R1 Influences R2"* implies that R1 affects R2 either positively (+) or negatively (-). A positive impact means that satisfaction of the source requirement (R1) can help satisfy the target requirement (R2). A negative impact implies that the requirements have a conflict with each other.
- Given two requirements R1 and R1.1, *"R1 Is a Parent for R1.1"* (or equivalently, *"R1.1 Is a Child for R1"*) implies that R1.1 refines R1. The following *Parent-Child* relationships can be defined between R1 and R1.1:
  - "R1 AND R1.1". This relationship implies that in order to satisfy R1, R1.1 has to be satisfied as well.
  - "R1 OR R1.1". This relationship implies that in order to satisfy R1, the satisfaction of R1.1 is sufficient but not necessary.

3.1.2. Conflict resolution algorithm

It is imperative that any conflicts among the requirements of the requirements model be resolved. The literature related to Requirements Engineering contains numerous works on conflict resolution among system requirements; the contributions reported in [36–39] are just a few of the examples. In [39], a conflict resolution process is proposed for resolving conflicts among the aspects identified during requirements engineering in the AspeCiS approach (An Aspect-oriented approach to Develop a Cooperative information System). In this process, dominant aspectual requirements are determined by computing a priority value for each aspect. We have used an adapted version of this method for resolving the conflicts among the requirements of the target methodology; this method has been selected because its underlying concepts are more consistent with our requirements metamodel.

A conflict occurs between two or more method requirements (leaf requirements in the requirements model) when these requirements introduce conflicting behavior in the target methodology; in other words, one requirement mandates the addition of a certain set of method fragments and relationships to the target methodology, while the other requirement results in the addition of method fragments and relationships which act in contradiction of the first set. In our proposed approach, the method engineer determines the conflicting requirements in the SF2R method base (based on the aforementioned definition). When a situational methodology is being constructed, these relationships are extracted (along with the requirements) from the method base by executing the transformation patterns. The resolution process involves the following sequence of stages:

1. Stakeholder Prioritization: In [39], the method proposed in [40] is used for this purpose. We use the same method in our proposed approach, as its concepts are consistent with the environments that are typically investigated for extracting the situational factors (i.e., organizations). In this method, three features are considered for prioritizing the stakeholders:
   - Power: A stakeholder affecting the decision making of other stakeholders has such a feature. The strength is taken from owning financial resources or equipment, or is based on the hierarchy of roles in the organization.
   - Legitimacy: This feature is attributed to stakeholders with legal, religious or supposed claims that affect the behavior, orientation or output of the organization.
   - Urgency: This feature is attributed to the stakeholders who are under one of the following conditions:
     - Their request must be met in a limited period of time.
     - Their request is important or critical to the stakeholder.

   In [39], a combination of the three attributes is used for prioritization. Prioritization is applied as follows. A stakeholder having just one of the above features is considered a *latent* stakeholder; this group of stakeholders is considered less important, and is assigned a low priority (a value of 1). A stakeholder with two of the above features is considered an *expectant* stakeholder, and is assigned a medium priority (a value of 2). A stakeholder with all three features is considered a *definitive* stakeholder, and is assigned a high-priority (a value of 3).

2. Conflict resolution: Using the concepts and methods introduced in [39], we propose the following algorithm for resolving the conflicts among methodology requirements in SME:
   2.1. A Stakeholder∗Requirement Matrix is formed in which the priorities of the stakeholders are first entered; then, the degree of importance of each requirement is determined based on the views of the stakeholders relevant[1] to (involved in) the requirement. Every stakeholder can assign a value of 1 (very low), 2 (low), 3 (average), 4 (high), or 5 (very high) to each of his/her related requirements. It should be noted that this matrix only contains the requirements that are involved in conflicts; furthermore, all of the requirements should be at the lowest level of the requirements hierarchy (i.e., as leaf requirements).
   2.2. The following four factors are computed for each requirement and added to the Stakeholder∗Requirement Matrix:
   - Priority of the Requirements: The priority of each requirement is computed based on the importance assigned to the requirement by its relevant stakeholders (specified in the previous step of the algorithm) and the

[1] Stakeholders are allowed to refrain from quantifying a situational factor, in case they find it irrelevant to the project at hand and its environment.

priorities of those stakeholders, according to Formula (1).

$$P(Rj) = \frac{\sum P(Si, j) * Vi, j}{\sum P(Si, j)} \quad (1)$$

Where $P(Rj)$ is the priority computed for requirement Rj, Si, j is the ith stakeholder involved in Rj, $P(Si,j)$ is the priority of $Si.j$, and Vi,j is the importance assigned to Rj by Si,j (as determined in the previous step).

- Sum of the priorities of the involved stakeholders: For each requirement, the sum of stakeholder priorities is computed according to Formula 2.

$$SW(Rj) = \sum P(Si, j) \quad (2)$$

Where $SW(Rj)$ is the sum of the priorities of the stakeholders involved in Rj.

- Number of involved stakeholders: For each requirement, the number of stakeholders involved in that requirement is determined.
- Number of conflicts: For each requirement, the number of conflicts that it has with other requirements is computed.

2.3. A Requirement∗Requirement Matrix (Conflict Matrix) is formed, which specifies the requirements that are in conflict. All of the requirements should be at the lowest level of the requirements hierarchy (i.e., as leaf requirements). The requirements in this matrix should first be ordered in such a manner that the less important requirements are examined first. The reason for this is that such a requirement may be involved in conflicts with more important requirements, and all of these conflicts will be eliminated by applying the conflict resolution strategy, resulting in a larger number of stakeholders being satisfied.
The requirements are arranged in ascending order of priority. If two requirements are equal in this sense, they are ordered based on the sum of the priorities of the stakeholders involved (in ascending order); if still equal, they are arranged based on the number of stakeholders involved (in ascending order); and if still equal, they are arranged based on their number of conflicts (in descending order). If two requirements are still equal after the above steps, they are ordered by the method engineer.
After ordering the requirements in the Conflict Matrix, the requirements model is used for filling the matrix: If there is a conflict between Ri and Rj, cell (i,j) will be marked; it should be noted that conflicts are only marked once, so an upper triangular matrix is produced.

2.4. The conflicts shown in the Conflict Matrix are resolved: For each pair of conflicting requirements, resolution is performed as follows:

- The requirement with the higher order in the matrix is designated as the dominant requirement, and is kept as is.
- Stakeholders are consulted and with their consent, the recessive requirement is replaced with another requirement that corresponds to the same situational factor. Choosing the alternative requirement is done with the intention that situational factors be satisfied to a desirable degree.

As an example of applying the conflict resolution algorithm, consider the following two situational factors: "Schedule constraints on the project" and "Developers' business knowledge". The requirements extracted for these factors are shown in Table 1 (this table is in fact a subset of the requirements model produced at the MFIM level). In the "Corresponding Requirements" column of the table, the requirements in parentheses have a parent-child relationship with the ones shown outside the parentheses. The numbers in the "Involved Stakeholders" column of the table (in parentheses) show the priority of each stakeholder.

The relationships among the requirements in the requirements model show a conflict between "Perform domain analysis in each iteration" and "Speed up implementation", which needs to be resolved. The Stakeholder∗Requirement matrix is shown in Table 2, and the Requirement * Requirement matrix is shown in Table 3. Between the two conflicting requirements, "Speed up implementation" is the dominant requirement. Therefore, "Perform domain analysis in each iteration" is replaced by "Incorporate risk management".

3.1.3. High-level model of target methodology

The overall structure of the methodology, including its high-level lifecycle, is created according to the scope of the target methodology. The method engineer should decide what type of methodology is better for the project situation at hand. She/he then sets a value for (quantifies) the "Scope the Methodology" requirement according to the environment and features of the project. This requirement is considered top-priority, and should be satisfied at the MFIM level. If the method engineer does not quantify the requirement, the Object Oriented Software Process (OOSP) framework [41] is by default set as the general lifecycle of the methodology. The methodology model that is produced is represented as a UML4SPM activity diagram [42]. In order to choose a suitable modeling language for expressing the methodology model, we examined six process modeling languages: SPEM 1.1 and SPEM 2.0 ([43], [44]), DiNitto *et al.*'s modeling language [43], PROMENADE [43], Chou's modeling language [43], and UML4SPM. These languages were evaluated based on the criteria introduced in [45]. Due to the superior features of UML4SPM, it was ultimately selected for application in our proposed approach. In order to provide a sense of the superior features of UML4SPM, the results of evaluating it based on four of the criteria are shown in Table 4. Examples of UML4SPM activity diagrams are provided in Section 4.

3.2. TIM level

At this level, the fine-grained constituents of the methodology are identified using the requirements model and the available method base of method fragments (R2MF). Modeling at this level is performed in several iterations; the Methodology model, represented as UML4SPM activity diagram(s), is thus gradually completed by the method engineer. The modeling is performed as follows: In each iteration, the set of requirements corresponding to a situational factor is fed to the Medini-QVT tool as input, and the method's fine-grained constituents are determined through the execution of transformation patterns. The method engineer can then refine the methodology model attained in each iteration.

The R2MF method base has been implemented in the Medini-QVT tool for extracting the method fragments corresponding to the identified requirements. Medini-QVT is a tool for (semi)automatic execution of model transformations, and implements the relational language of QVT [9]. This tool has been used for implementing the transformation patterns, the SF2R method base, and the R2MF method base. It thus enables the automation of transformations for SME purposes.

3.3. TSM level

At this level, the TIM-level methodology model is fed to the tool as input, and for each of the tasks in the model, the corresponding concrete technique(s) for performing the task (if available) are extracted from the R2MF method base and added to the

Table 1
Requirements corresponding to sample situational factors.

Situational factor	Involved stakeholders	Corresponding requirements
Schedule constraints on the project	Programmer (1)	Break down into smaller tasks, Limit modeling, Constrain implementation of requirements (Prioritize requirements), Evaluate alternative hardware devices and select the fastest, Speed up implementation, Limit changes
	Technical Manager (2) Domain Expert (3) Analyst (2)	
Developers' business knowledge	Programmer (1)	Iterative-Incremental Lifecycle (Perform domain analysis in each iteration, Apply repetitive validation),
	Technical Manager (2) Domain Expert (3) Analyst (4)	
		Incorporate risk management, Use people who are familiar with the business area (Use JAD sessions, Use ambassador users), Improve business knowledge of team members

Table 2
Sample stakeholder∗requirement matrix.

Requirement		Speed up implementation	Perform domain analysis in each iteration
Stakeholder	Priority		
Programmer	1	5	3
Technical manager	2	4	2
Domain expert	3	3	4
Analyst	2	2	3
Requirement priority		26.8	25.8
Total priorities of involved stakeholders		8	8
Number of involved stakeholders		4	4
Number of conflicts with other requirements		1	1

Table 3
Sample requirement∗requirement matrix.

Requirement	Perform domain analysis in each iteration	Speed up implementation
Perform domain analysis in each iteration		×
Speed up implementation		

methodology model by execution of the transformation patterns. There is also the possibility of refining the methodology model at this level. The resulting methodology model can be enacted in the real development environment, and can be polished as needed at the method engineer's discretion.

3.4. SF2R method base

The SF2R method base is used for mapping situational factors to methodology requirements. The fragments in the method base are of the following format:

< (quantified situational factor), (requirement related to situational factor (requirement related to requirement and …), …) >.

The above structure denotes that a quantified situational factor is satisfied by the requirements in the second component, and the requirements can be recursively refined into other requirements. For example:

< (Schedule constraints on the project), (Break down into smaller tasks, Limit modeling, Constrain implementation of requirements (Prioritize requirements)>.

Fig. 3 shows the metamodel of this method base.

The concepts depicted in Fig. 3 are the same as those shown in Fig. 1. In total, 15 situational factors have been defined in this method base; these factors will be explained in Section 5.4. Also, the requirements related to each situational factor and the relationships among them (Refinement, Dependency, Parent-Child, and Influence) have been added to the method base. A partial view of the contents of the SF2R method base is shown in Fig. 4 (in Medini-QVT); the 'Properties' pane contains the attributes of the "Prioritize Requirements" requirement, and shows that this requirement refines the "Financial Constraints on the Project" situational factor.

Table 4
Partial results of evaluating UML4SPM.

Assessment criterion	Evaluation result
Semantic richness (Expressiveness)*	UML4SPM provides the notation for modeling the key elements of software development methodologies according to the SPEM metamodel.
Consistency of notation	UML4SPM models are notationally consistent.
Clarity of notation	UML4SPM notation can be understood easily, especially by modelers who are familiar with UML.
Extensibility of concepts and notation**	New concepts can be added to the metamodel of UML4SPM through the use of the extension mechanisms provided by UML.

Explanations:
* Does the language fully express what is actually performed and involved in the enactment of software development methodologies?
** Is it possible to extend the metamodel of the language in order to allow the modeling of new concepts related to software development methodologies?

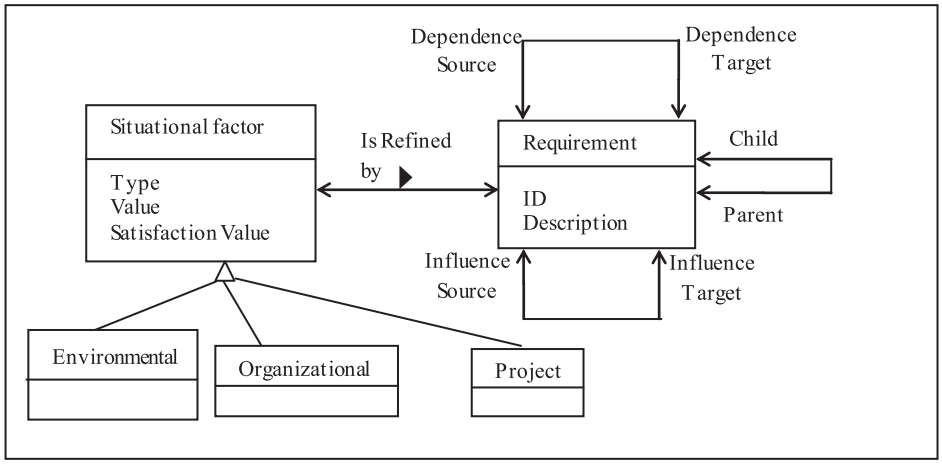

Fig. 3. Metamodel of the SF2R method base.

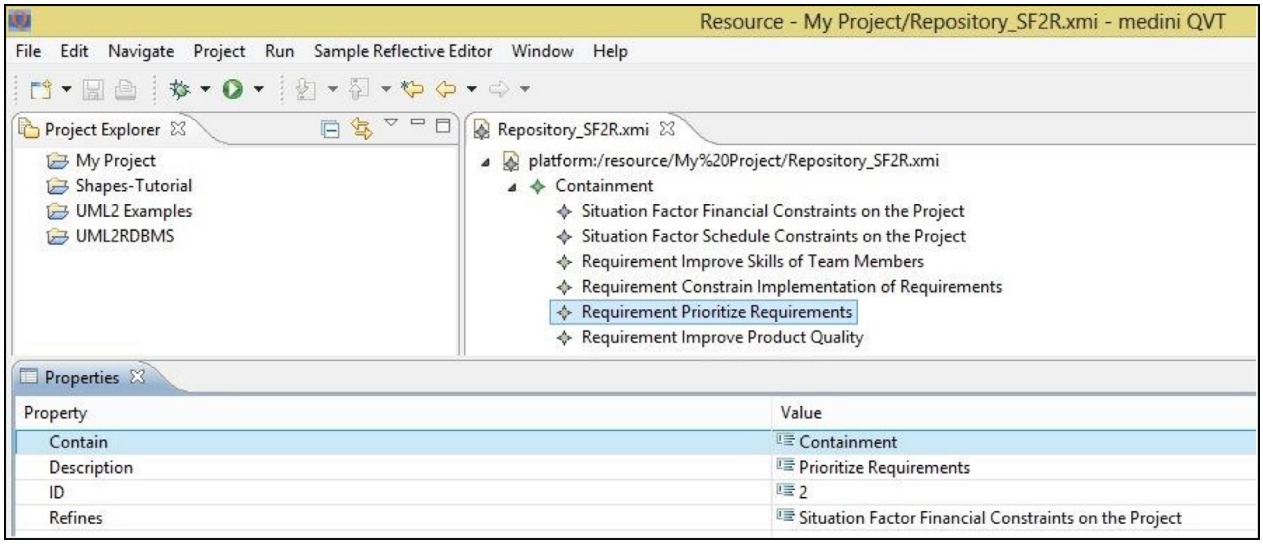

Fig. 4. Partial view of the contents of the SF2R method base.

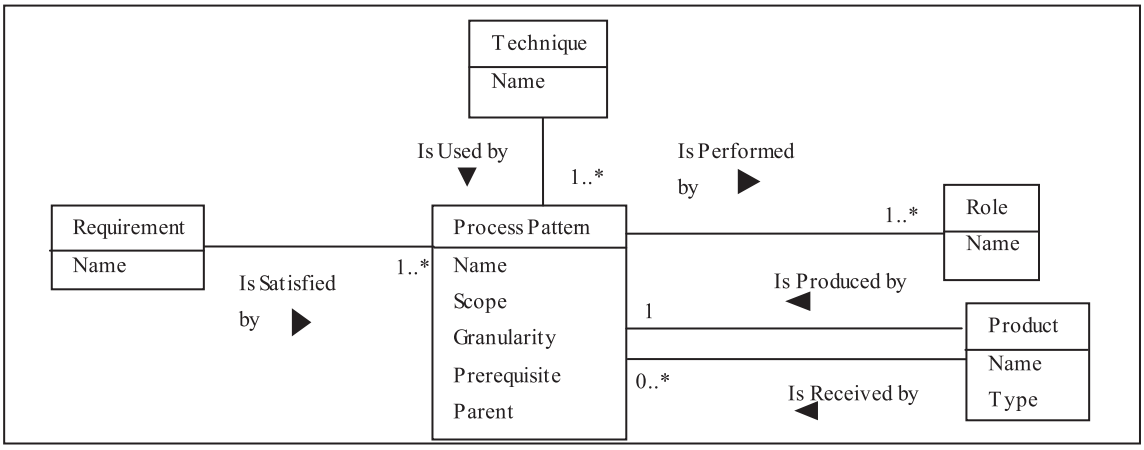

Fig. 5. Metamodel of the R2MF method base.

3.5. R2MF method base

The R2MF method base is used as a source of method fragments for satisfying the requirements at different levels of the proposed framework. The metamodel of this method base is shown in Fig. 5. We have adopted SPEM 2.0 (Software and System Process Engineering Metamodel) [46] for the definition of this method base. In SPEM 2.0, the Method Content package defines the core concepts for specifying basic method contents. There are three types of elements in this package: roles, work products, and tasks.

SPEM 2.0 incorporates the notions of lifecycle, phase, activity, task, and technique for defining the lifecycle of a process, depicting different levels of abstraction and constraining the order in which activities can be performed. These concepts have been followed in the definition of the R2MF method base.

Process Patterns are the main method fragments: These patterns are in fact software development *work units* (high-level *phases*, intermediate-level *activities*, and low-level *tasks*) that have been proven successful in practice. These patterns have been chosen from among the process patterns already utilized in software engineering; i.e., the process patterns that we use in our approach are Software Engineering (SE) work units which have been proved successful in practice (this is somewhat different from SME process patterns [47], which have been extracted from existing SME approaches). Process patterns are complemented with other fragments: Each process pattern is associated with certain Roles (people involved), Products (results produced), and Techniques (concrete methods for implementing the pattern). It is important to distinguish between process patterns and transformation patterns: transformation patterns are transformation rules which are executed to extract process patterns from the R2MF method base.

The method fragments of the R2MF method base have been extracted from various sources: the fragments related to agile methodologies have been derived from those proposed in [48]; the phases and high level activities related to other types of development approaches, including Model-Driven Development [49], Component-Based Development [50], Aspect-Oriented Development [51], High Integrity Systems Engineering [52], and Web Engineering [53] have also been added to this method base. Each of these resources [48–53] introduces process patterns for a specific methodology type (according to a particular domain, paradigm, or development approach), and creates a high-level software development process which can potentially be used for instantiating a situational methodology of that particular type; however, the instantiation process is not elaborated upon. In comparison, our approach is generic (type-independent), and provides a comprehensive model-driven step-by-step process for constructing the target situational methodology.

In total, around 330 fragments have been defined in R2MF. In addition, the relationships that exist between process patterns at the "phase" and "activity" levels have been added. In contrast, the relationships between "task" process patterns are specified in the rules that accompany the transformation patterns (explained in Section 4.2). The reason for this is that high-level "phases" and "activities" are somewhat predetermined for a specific methodology type, and are therefore fixed when the methodology type is selected, whereas lower-level "tasks" are determined according to the requirements of the target methodology.

The method engineer revisits the SF2R and R2MF method bases iteratively and refines them as needed (by adding new situational factors and method fragments, and defining new value ranges for existing situational factors). It should be noted that the changes applied to the method bases do not affect the transformation patterns.

The attributes of the "Process Pattern" class in Fig. 5 are as follows:

- **Name**: The unique name of the process pattern.
- **Scope**: The application scope of the process pattern; this feature may consist of one or more development approach types (Agile, Model-Driven, Component-Based, etc.).
- **Granularity**: This feature denotes the granularity of the process pattern, and can have one of three possible values: Phase, Activity, or Task. Phases are at the highest level, and may consist of activities and/or tasks. Activities may consist of tasks and/or other (nested) activities. Tasks are atomic and reside at the bottom level of the hierarchy.
- **Prerequisite**: The prerequisite patterns (work units) of the process pattern.
- **Parent**: The coarse-grained pattern (phase or activity) that contains the process pattern as a constituent.

A partial view of the contents of the R2MF method base is shown in Fig. 6. The 'Properties' pane contains the attributes of the "Fine-Tune the Methodology" process pattern, and shows that this pattern is performed by the "Project Community" role, produces "Refined Methodology" as a product, and satisfies the "Improve Quality of Used Methods" and "Flexibility" requirements.

4. Proposed process and transformation patterns for applying PBMDD4SME framework

In this section, a process is provided for conducting SME based on the proposed framework. Moreover, the transformation patterns used for this purpose will be explained. It should be noted that the models created in our proposed approach comply with OMG's view on modeling, as we use UML class and object diagrams for structural modeling, and UML4SPM for software process modeling. Thus, all the models created in the proposed approach are MOF-compliant.

4.1. Proposed model-driven SME process

At the first level of the proposed framework, a Paradigm-based approach is used for determining the overall structure of the target methodology. At the next levels, an Assembly-based approach is applied to complete the model. Fig. 7 shows the steps of the first two stages of the proposed process (at the MFIM level). In order to better clarify the various stages, an example of enacting the process will be presented at the end of this section.

The stages of the process are as follows:

1. A set of situational factors are extracted for the situation of the project at hand. Each of the situational factors has a value range that is determined by the method engineer. The situational factors are then given to stakeholders, who assign a value to each situational factor from its corresponding value range according to the project circumstances; the method engineer then documents the stakeholder(s) from whom the information has been elicited. Completion of the requirements model typically requires several iterations, and the method engineer's expertise is essential for this task. The sub-stages of this stage are as follows:
   1.1 An unprocessed situational factor is chosen as the target factor.
   1.2. The requirements related to the target situational factor are extracted from the SF2R method base. This is automatically done based on the value of the situational factor and through the execution of transformation patterns in the tool. Extracted requirements can be refined by the method engineer.
   1.3. A subset of the requirements extracted for the target situational factor is determined for satisfaction; this subset should provide the maximum Satisfaction Value (SV) for the target factor.
   1.4. If there remains an unprocessed situational factor, return to step 1-1.
   1.5. After the determination of the requirements related to each situational factor, the transformation patterns and the SF2R method base are used for defining the relationships among requirements. The requirements model created so far is considered as an input to the transformation patterns. Upon

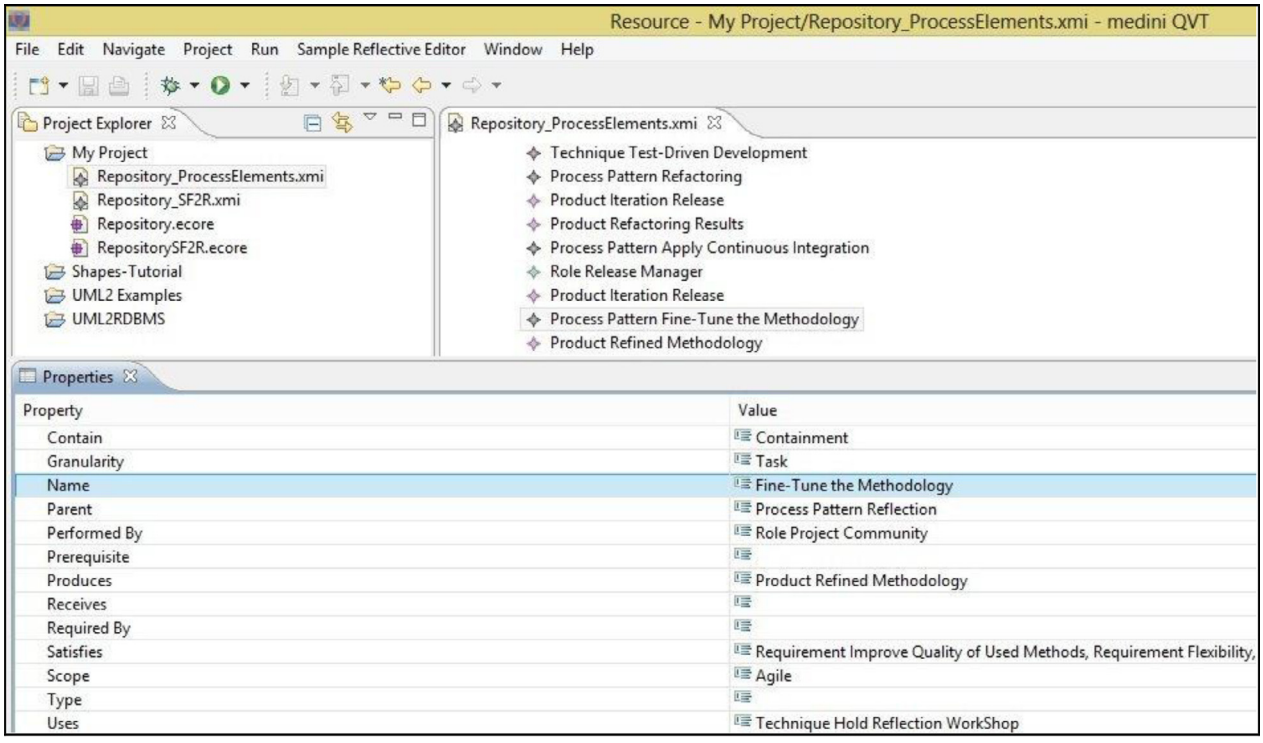

Fig. 6. Partial view of the contents of the R2MF method base.

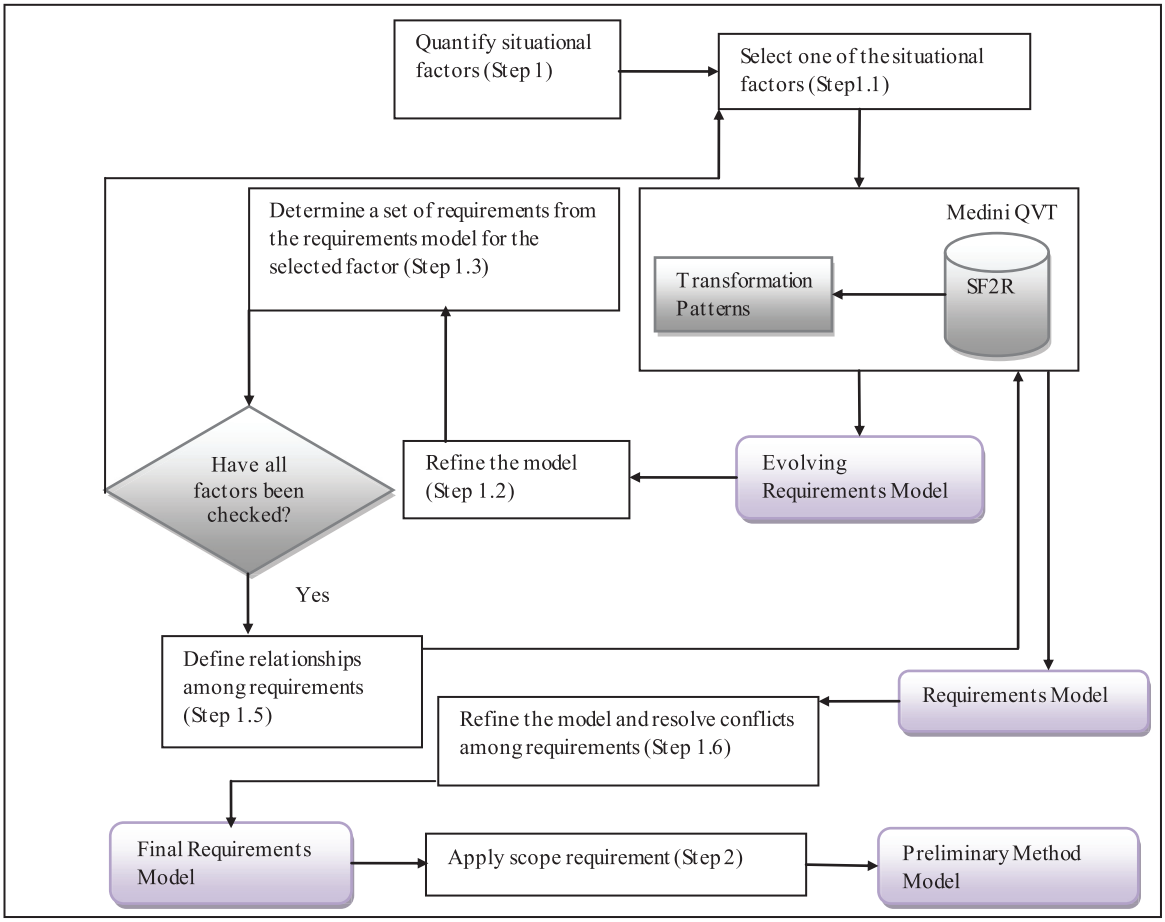

Fig. 7. Proposed model-driven method engineering process at the MFIM level.

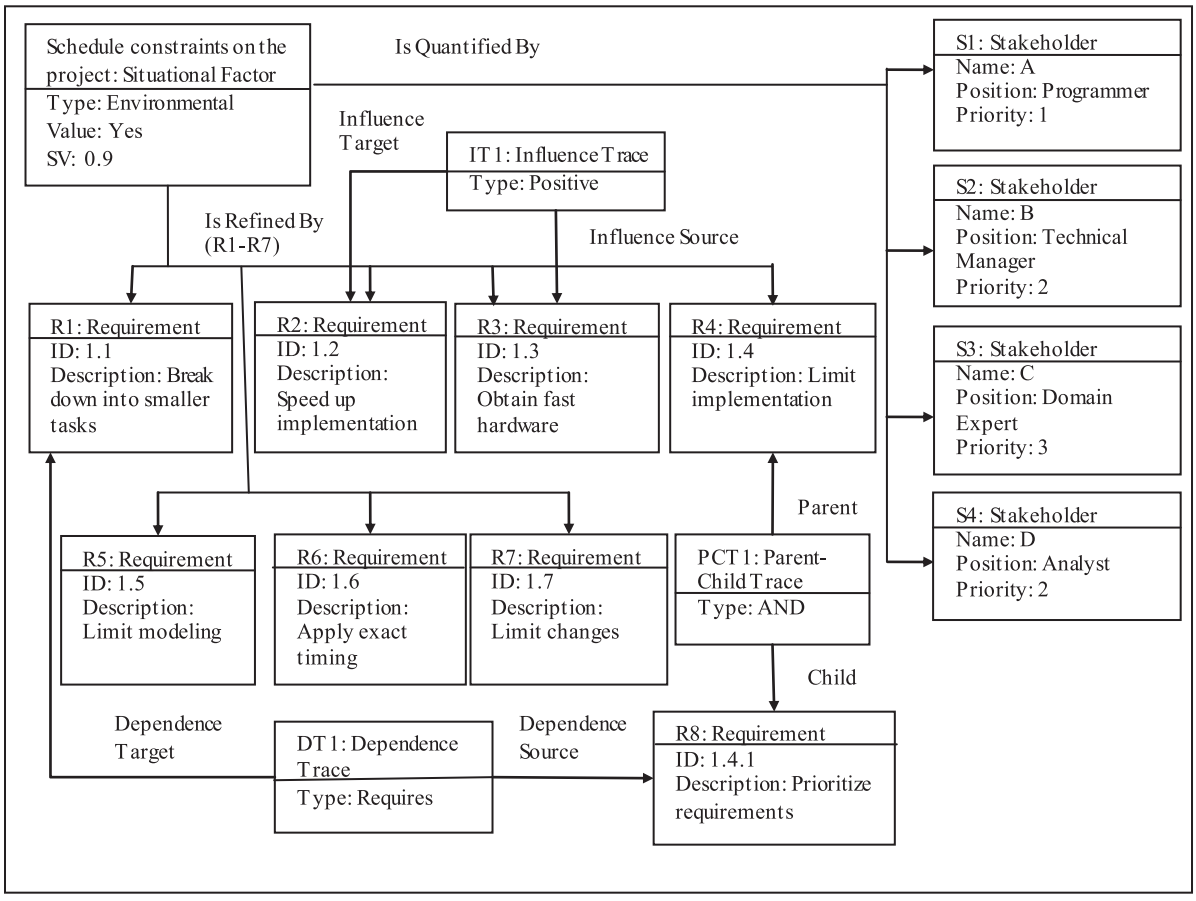

Fig. 8. Example of a requirements model (for the "Schedule constraints of the project" factor).

execution, these patterns extract all the relevant relationships among the requirements from the SF2R method base, and add them to the requirements model.

1.6. The method engineer reviews the output model and refines it as needed. If conflicts are detected among the requirements, they are resolved through applying the conflict resolution algorithm.

As an example, a partial requirements model, related to the "Schedule constraints on the project" factor, is shown in Fig. 8. If the method engineer feels the need to change the value range of a situational factor or add a new situational factor, he/she can execute the transformation patterns for those situational factors in order to apply the required changes; new requirements may thus be added to the requirements model.

2. The method engineer specifies a value for the "Scope the Methodology" requirement according to the environment and features of the project, and thus determines the methodology type (this is considered the most important requirement and should be applied before other requirements). High-level phases and activities (including their relationships and umbrella activities) have already been defined in the R2MF method base for different methodology scopes (types), and are extracted from the method base by executing the transformation patterns based on the value determined for the scope. This stage is performed automatically by the tool. An example of the produced model is shown in Section 5.4.

Fig. 9 shows the steps of the last two stages of the proposed process (at the TIM and TSM levels).

The stages are explained below:

3. Finer-grained constituents of the methodology (at the TIM level) are extracted based on the requirements model. Firstly, an unprocessed situational factor is chosen. The extraction of method fragments from the R2MF method base is performed iteratively, for each situational factor selected, through the following sub-stages:

3.1. A subset of the requirements that provides the maximum SV for the selected factor has been determined in previous stages. For each requirement R in this set, the following stages will be applied (depending on R's relationships with other requirements):

3.1.1. If R has an 'Overlaps' relationship with requirement(s) Ri (i = 1 … n), Ri are marked as reviewed and will not be processed further.

3.1.2. If R has a 'Requires' relationship with requirement(s) Ri (i= 1 … n) (i.e., Ri are prerequisites for R), fragments should first be extracted for Ri; therefore, stages 3-1-1 and 3-1-2 will first be applied to Ri.

3.2. For each of the requirements, the corresponding method fragments (process patterns, roles, and input/output products) are determined and entered into the methodology model. This task is performed based on the R2MF method base and through the execution of the transformation patterns in the tool. The relationships between task process patterns are determined through executing the transformation patterns (explained in Section 4.2).

3.3. A method engineer verifies and refines the methodology model created so far.

3.4. If there remains an unprocessed situational factor, go back to Step 3-1.

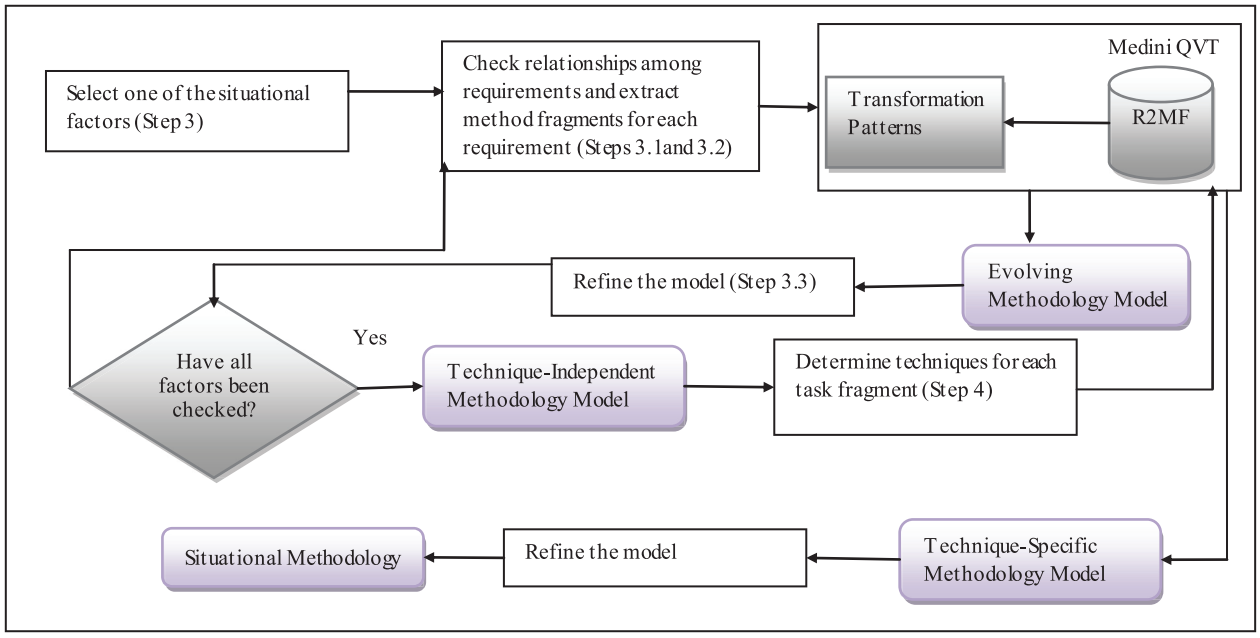

Fig. 9. Proposed model-driven method engineering process at the TIM and TSM levels.

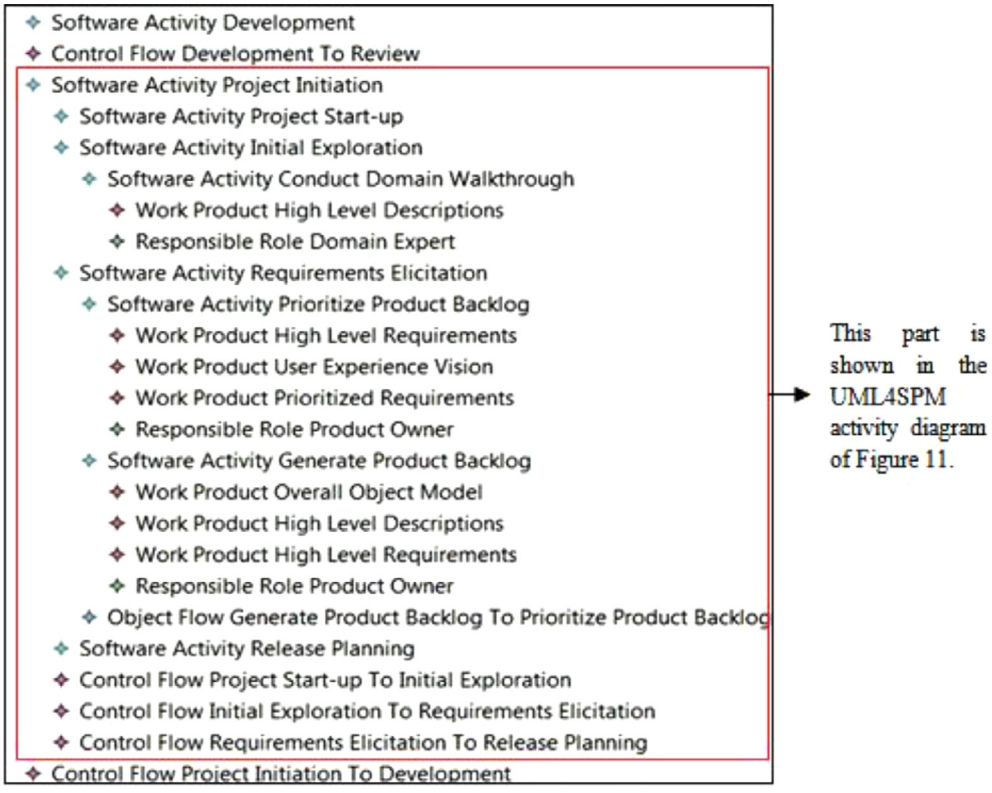

Fig. 10. Sample output generated by Medini-QVT: Result of execution of transformation patterns at the TIM level.

An example of the output generated by Medini-QVT, as a result of executing the transformation patterns at the TIM level, is shown in Figs. 10 and 11 shows the produced methodology as a UML4SPM activity diagram. If the SF2R method base is updated during this stage, the method engineer should re-execute the transformation patterns related to the previous stage to update the requirements model. Then, the methodology model should be updated by re-executing the current stage's transformation patterns on R2MF. Thus, returning to the previous stage is a regular occurrence.

4. The output model of the previous stage (TIM-level) depicts the purpose and detailed structure of the methodology, but not how to enact its activities. To add this level of detail, the model is input to the tool, and techniques are determined for each task based on the R2MF method base and through execution of the relevant transformation patterns in the tool. The output is the TSM-level methodology model. The method engineer verifies the model and refines it if necessary. The produced methodology is then enacted in the real world and is further

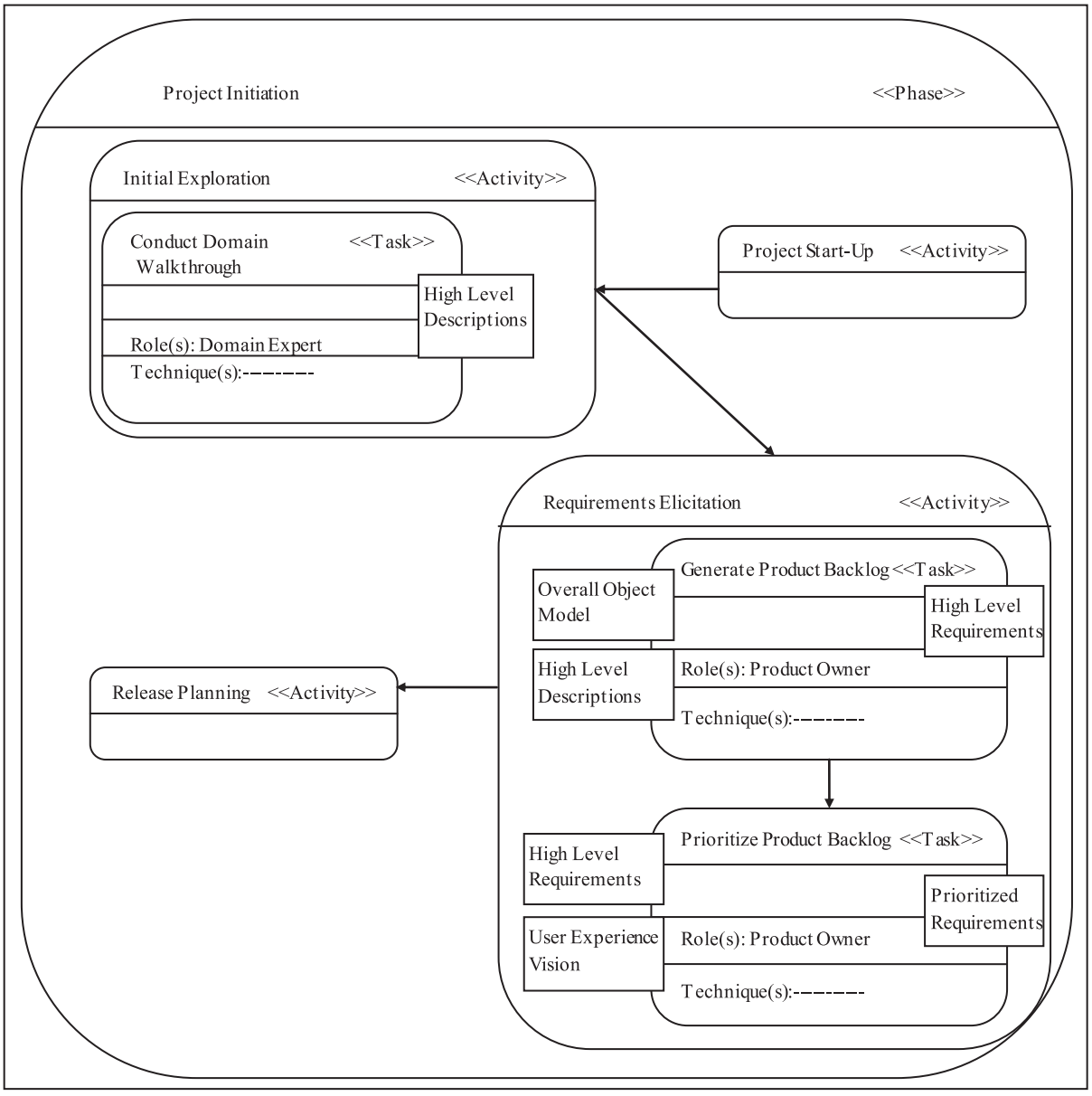

Fig. 11. Partial model of the "Project Initiation" phase: Result of execution of transformation patterns at the TIM level.

refined by the method engineer. An example of the output generated by Medini-QVT as a result of executing the transformation patterns at the TSM level is shown in Fig. 12. If the SF2R or R2MF method bases are changed during this stage, it is possible to return to previous stages and re-execute the transformation patterns, and thus apply the required changes to the methodology model.

An example of applying the proposed SME process on two situational factors is shown in Fig. 13.

4.2. Transformation patterns

In this section, the transformations necessary for producing the various models of the target methodology are proposed as 13 transformation patterns. The general idea of these patterns is taken from [10]. High-level definitions of these patterns are provided in Table 5. All of the patterns in this table are transformation patterns, some of which are used for extracting process patterns from the R2MF method base.

We have also used a template for describing these patterns in more detail; this description template provides the pattern's name, problem definition, solution definition, expression of the pattern in QVT, and an example of the pattern's application; as an example, Table 6 shows the detailed description of the "Add techniques" pattern through the use of this template. Due to space limitations, we will not provide the detailed descriptions of the rest of the patterns herein.

In Table 6, Part 1 checks if there are any techniques in the R2MF method base that are related to a process pattern (work unit) in the methodology model produced so far. The process pattern itself is specified in Part 2. In Part 3, the identified techniques are added to the process pattern. In fact, Parts 1 and 2 constitute the 'before' state of the pattern, and Part 3 denotes the 'after' state.

It should be noted that the proposed transformation patterns may involve vertical as well as horizontal transformations, in that:

- According to the proposed SME process, the methodology model produced at each level is completed gradually with the constant involvement of the method engineer. Therefore,

Table 5
General descriptions for proposed transformation patterns.

Transformation pattern name	Transformation pattern description	
	Problem	Solution
Identify the requirements corresponding to a given situational factor	Situational factors are too abstract (methodology-independent) and need to be refined.	Based on the refinement relationships between situational factors and requirements (defined in the metamodel of the SF2R method base), the requirements corresponding to each situational factor are identified and added to the requirements model.
Identify parent-child relationships	Certain requirements in the requirements model are coarse-grained and need to be broken down into finer-grained requirements.	The SF2R method base is used for identifying fine-grained requirements that have a parent-child relationship with the coarse-grained requirements. The child requirements thus identified are added to the requirements model under their coarse-grained parents.
Identify dependency relationships among requirements	It is possible for 'overlapping' or 'prerequisite' dependencies to exist among requirements.	The SF2R method base is used as a basis for identifying these dependency relationships. Identified relationships are added to the requirements model.
Identify influence relationships among requirements	Requirements may influence one another. These effects may be positive (+) or negative (-).	Based on the influence relationships defined in the metamodel of the SF2R method base, these relationships are identified and added to the requirements model.
Extract general structure for target methodology	A general structure should be determined for the target methodology, depending on its scope.	The general structure is determined by extracting high-level phases/activities (including umbrella activities) from the R2MF method base based on the value assigned to the scope of the target methodology. The transitions among phases/activities are also determined.
Map requirements to process patterns (work units)	For each requirement, the process patterns that satisfy them need to be extracted from the R2MF method base.	According to the 'satisfy' relationships defined in the metamodel of the method base, suitable process patterns (work units) are extracted for each requirement, and are then added to the methodology model.
Identify prerequisite process patterns (work units)	The process patterns (work units) extracted from the R2MF method base may have prerequisite process patterns that have not been considered.	The R2MF method base is used as a basis for identifying the prerequisites of each process pattern; these prerequisites (work units) are then added to the methodology model.
Add roles to work units	After the elicitation of work units (and prerequisites), suitable roles must be defined for performing them.	By using the 'perform' relationship defined in the metamodel of the R2MF method base, the roles corresponding to each work unit are identified and added to the methodology model.
Add input and output products to work units	After the extraction of work units (and prerequisites), input and output products must be determined for each work unit.	According to the relationships defined among the products and method fragments in the metamodel of the R2MF method base, the input/output products of the work units are extracted and added to the methodology model.
Determine transition relationships	By applying the previous transformation patterns, work units are added to the methodology model, but no transition relationships are defined among them.	Identification of transitions is based on the input and output products of the corresponding work units: If the output product of one work unit is the input to another, and both of them reside in the same parent work unit, a transition is established between them as an Object Flow.
Define parallel relationships	Some work units can be executed in parallel.	Parallel relationships are established among the work units if they 1) have the same common parent (reside in the same phase/activity), and 2) either do not have any prerequisites or share the same prerequisites. It should be noted that these two conditions are necessary, but not sufficient: There might be work units that cannot be executed in parallel even though they satisfy the above conditions; the method engineer will have to define a sequence of execution for such tasks.
Define synchronization relationships	There are work units with two or more prerequisites, all of which should be executed before the tasks can start.	A Join Node is added to the methodology model to depict the required synchronization: The prerequisite work units are designated as inputs to the Join Node, and the target work unit is designated as the output.
Add techniques	Techniques should be determined for each work unit in order to specify how the work units should be implemented.	According to the relationships defined between the work units and their related techniques (in the metamodel of the R2MF method base), the techniques associated with the work units of the methodology model are identified and added to the model.

Table 6
Detailed description for the "add techniques" pattern.

Pattern name	Add techniques
Problem	Work units have been added to the model, but their techniques have not been specified.
Solution	According to the relationships defined between tasks and their related techniques in the metamodel of the R2MF method base, the techniques associated with the work units of the methodology model are identified and added to the methodology model.
QVT Definition	**top relation** IdentifyTechniques{ varname1:String; ;varname2:String; ; varname3:String; ;varname4:String; Part 1 → **checkonly domain** left1 t: Repository::Technique{ name = varname1, UsedBy = pp : Repository::ProcessPattern { Granularity = varname2, name = varname3, Scope = varname4 } }; Part 2 → **checkonly domain** left2 SA: Simple_UML4SPM::SoftwareActivity{ Description = varname3, Kind =varname2 }; Part 3 → **enforce domain** right t1: Simple_UML4SPM::Technique{ Description = varname1, UsedBy = SA : Simple_UML4SPM::SoftwareActivity {}, Contain = SA }; }
Example	By applying this pattern to the methodology model for the "Architectural design" task, techniques such as "Early design", "Create system metaphor", "Hold JAD session" and "Explore alternative methodologies" are added to the methodology model.

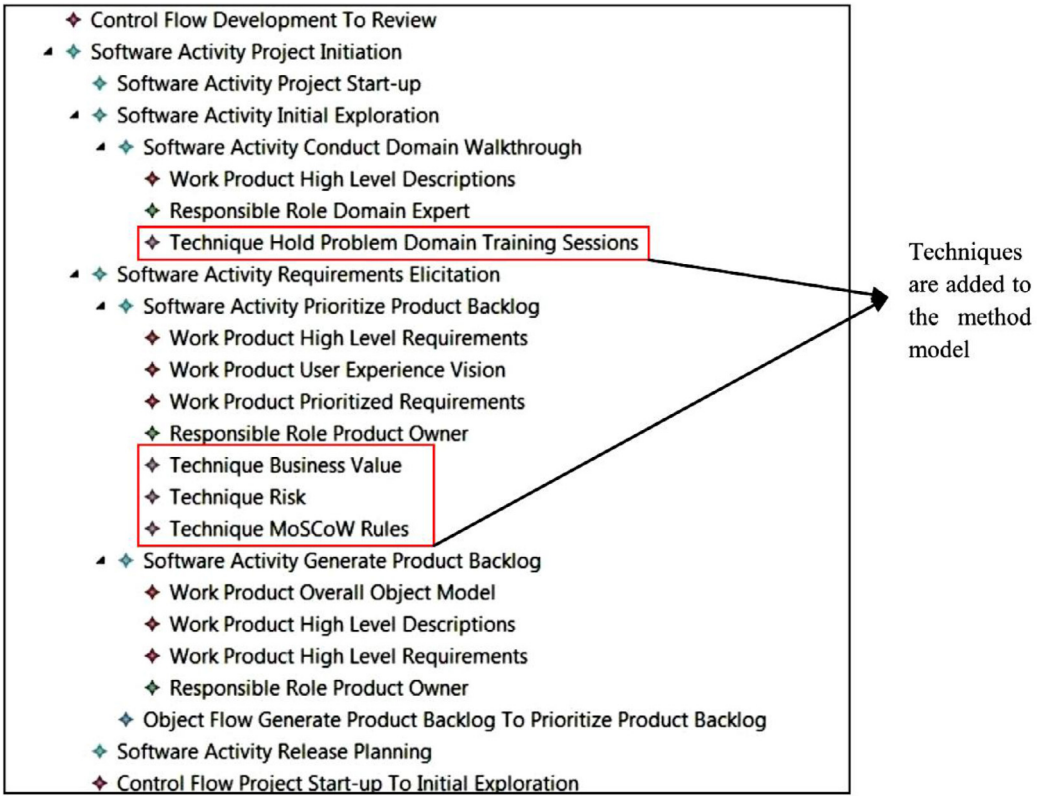

Fig. 12. Sample output generated by Medini-QVT: Result of execution of transformation patterns at the TSM level.

horizontal transformations are typically performed continuously at each level.
- Vertical transformations are evident between adjacent levels, as the output of each level is input to the next level down the hierarchy. Additional information is added along the way, which puts the resulting model at a lower level of abstraction.

5. Evaluation of proposed approach

In this section, PBMDD4SME has been evaluated through 1) comparison with other MDD frameworks used for SME purposes, and 2) comparison with other SME approaches. In addition, PBMDD4SME and its process have been applied to a real-world project, the results of which are analyzed later in this section.

5.1. Comparison of PBMDD4SME with other MDD frameworks used in SME

Unfortunately, a comprehensive set of criteria for evaluating MDD frameworks is not available. This comparison is therefore carried out based on a set of criteria that every MDD framework should satisfy [5]. Brief explanations of these criteria are given below:

- Definition of modeling levels: Each of the modeling levels defined in the framework is at a specific abstraction level. This criterion refers to the viewpoint abstraction provided at each level.
- Transformation Type: The transformation rules defined for converting models to one another can be vertical or horizontal. Vertical transformation when the source and destination models are at different levels of abstraction, while horizontal transformation is used for converting models that are at the same abstraction level.
- Potential for automation of problem-to-solution transformations: This refers to the level of automation in executing the transformations defined in the framework. The possible values for automation potential are as follows:
  - Low: Transformations have been addressed in the framework, but the framework does not provide methods for producing solution-domain models from problem-domain models.
  - Medium: The framework provides methods for producing solution-domain models from problem-domain models, but these methods cannot be performed (semi)automatically.
  - Medium to High: The framework defines transformations that can be applied (semi)automatically, but it does not provide a method base of method fragments for this purpose.
  - High: The framework defines transformations that can be applied (semi)automatically. Moreover, it also provides a method base of method fragments for this purpose.
- Portability: This criterion signifies the concept(s) that differentiate the modeling levels defined in the framework. Portability is supported for higher-level models across the concepts representing (introduced at) the lower levels.

The results of this evaluation are shown in Table 7. Parts of the table have been reproduced from [5]. It should be noted that all of these MDD frameworks, except for MDA, are specifically intended for SME purposes; MDA has been added as a reference point, due to its generality of purpose. It can be observed that the main strength of our proposed approach (PBMDD4SME) lies in its high level of automation in performing model transformations. Also, the modeling levels defined in the framework, and the degree of abstraction that they provide, enhances the reusability and portability of the models.

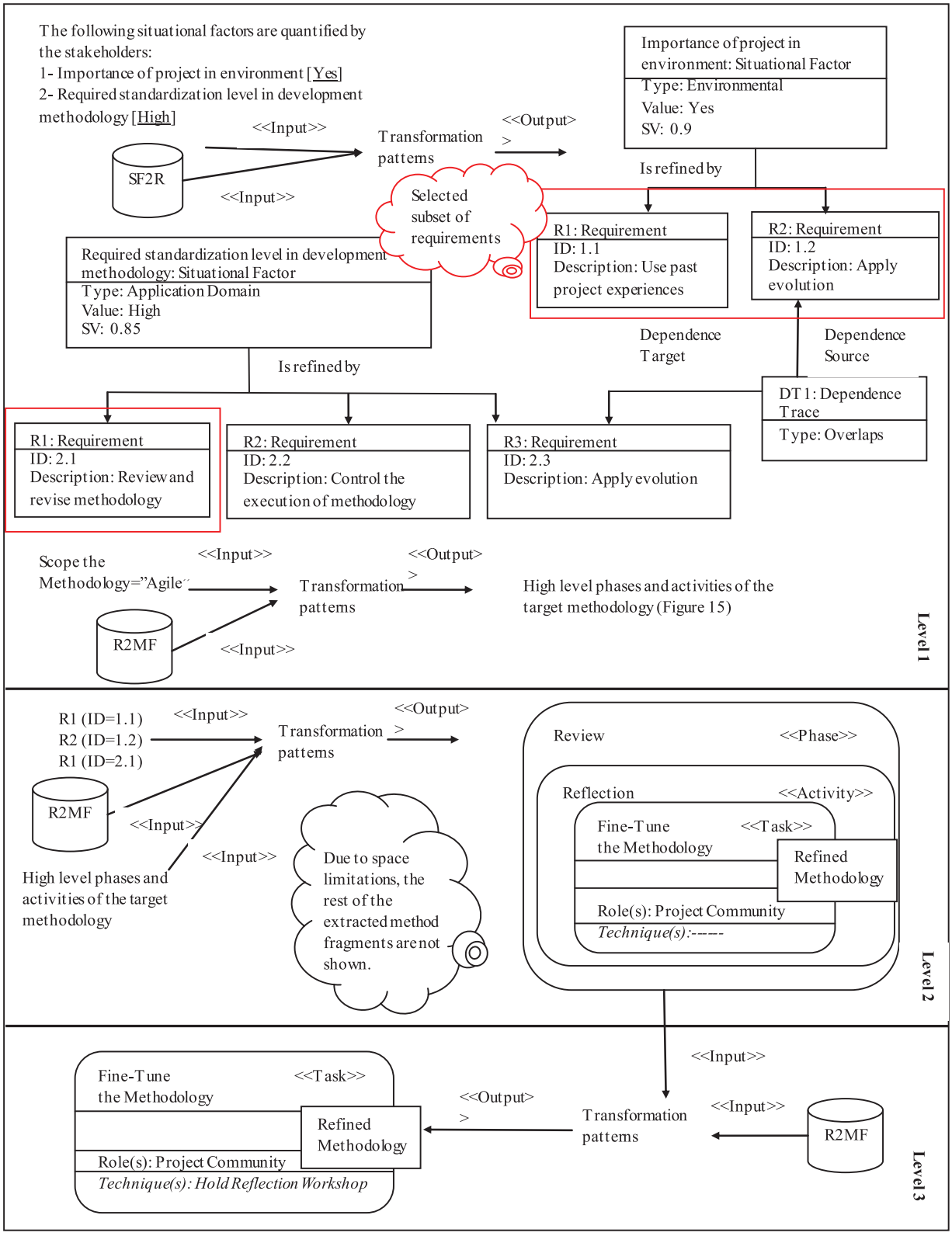

Fig. 13. Example of application of proposed process.

5.2. Comparison of PBMDD4SME with other SME approaches

This comparison is carried out based on the set of criteria proposed in [5] for evaluating SME processes. These criteria are the results of adapting the process evaluation criteria defined for SE to the SME context. Brief explanations of these criteria are given below:

- Design model: The model(s) created throughout the SME process.

Table 7
PBMDD4SME framework in comparison to other mdd frameworks.

MDD framework						
Criterion	MDA	Model driven process engineering [17]	MDE approach to software process tailoring [18]	MDSME [5]	Methodological framework [16]	PBMDD4SME
Definition of modeling levels	Viewpoint abstraction: • Business viewpoint • System viewpoint • Software viewpoint	Linguistic Metamodeling	Viewpoint abstraction: • Requirements-independent method model • Requirements-specific method model	Viewpoint abstraction: • Enactment-independent viewpoint • Paradigm-independent viewpoint • Paradigm-specific viewpoint • Platform-specific viewpoint	Viewpoint abstraction: • Technology- and tool-independent viewpoint • Technology- and tool-specific viewpoint	Viewpoint abstraction: • Fine-grained method-fragments-independent viewpoint • Technique-independent viewpoint • Technique-specific viewpoint
Transformation Type	• Vertical • Horizontal	• Vertical • Horizontal	• Vertical	• Vertical • Horizontal	• Vertical • Horizontal	• Vertical • Horizontal
Potential for automation of problem-to-solution transformations	Low (based on [54])	N/A (Transformation of models has not been addressed)	Medium to High	Medium	Medium to High	High
Portability as to:	• Platforms	• Method execution environments	• Project situations	• Method platforms • Project situations	• Technology and tools	• Techniques • Project situations

- Potential for process automation: What level of automation is provided for performing the SME process? The possible values for this criterion are as follows:
  - Low: Most of the process is performed manually.
  - Medium: A part of the process (transformation rules) is performed automatically, but extraction of method fragments needs manual intervention by the method engineer.
  - High: Most of the process is performed automatically by providing method bases of method fragments and automatic execution of transformation rules.
- Portability of methodology model: This criterion signifies the concept(s) across which the methodology model is portable. The possible values for this criterion are as follows:
  - Low: Portability of the methodology model is not addressed.
  - Medium: There is portability as to certain concepts, but these concepts do not cover the full set of SME concepts.
  - Medium to High: There is portability as to the basic concepts of the SME context (such as project situations).
- Complexity management: Structural complexity (such as complexity of design models) and behavioral complexity (such as complexity in performing the steps of the SME process) can be managed by mechanisms such as selecting a suitable PML and providing an appropriate level of automation in performing SME process steps. The possible values for this criterion are as follows:
  - Low: Complexity management has not been explicitly addressed.
  - Medium: There are methods for complexity management in the proposed approach, but unmanaged complexity is still observable.
  - High: Complexity is fully managed (structural and behavioral).
- Maintainability: Does the process facilitate model change? The possible values for this criterion are as follows:
  - Low: Any change in any part of the methodology models leads to a ripple effect throughout the models.
  - Medium to High: Mechanisms such as intermediate models are provided for enhancing the maintainability of the methodology models.
  - High: In addition to intermediate models, automatic propagation of maintenance changes to the methodology models is supported.
- Environment and tool dependency: Performing SME process steps may require specific environment(s) or tool(s). The possible values for this criterion are as follows:
  - Low: There is no dependency to any specific tool or environment.
  - Medium: A specific environment is required with certain SME facilities (such as a method base of method fragments).
  - High: Specific SME environment(s) and tool(s) are required for performing the steps of the process.

It should be noted that only SME approaches that provide a crisply defined process for creating a situational method have been selected for comparison. The results of this evaluation are shown in Table 8. Parts of the table containing the results of evaluating other SME approaches have been reproduced from [5]. It can be observed that one of the main advantages of our proposed process is that it provides a method base of reusable method fragments for assembling the target methodology. Also, the modeling levels of the framework enhance the maintainability of methodology models, and facilitate complexity management.

5.3. Evaluation of transformation patterns

In [8], a set of evaluation criteria has been proposed for comparing transformation languages and tools. Based on these criteria and those proposed in [55–57], a set of criteria is proposed for evaluating the proposed transformation patterns. These criteria are briefly explained below:

Table 8
PBMDD4SME framework in comparison to other SME approaches.

SME approach				
Criterion	Generic process for SME [17]	MEMA-model [19]	Method engineering from MRS [20]	PBMDD4SME
Design model	N/A	Semi-open method	Decision metamodel	Models: • Fine-grained method-fragments-independent • Technique-independent • Technique-specific
Potential for process automation	Low: • Selection and use of method fragments	N/A	Medium: • Use of a CAME tool for method engineering	High: • Use of a tool to execute transformations
Portability of methodology model	Low: • Limited to the paradigm-based approach	Medium to High: • towards project situations	Medium: • towards relation types • towards detailed descriptions	Medium to High: • towards techniques • towards situations
Complexity management	Low	Medium	Medium	High: • Automatic execution of a large part of the process
Maintainability	Low: • Direct mapping of requirements to method fragments	Medium to High: • Use of design models	High: • Use of a meta- modeling approach • Appropriate level of automation through the use of the tool	High: • Appropriate level of automation through the use of the tool • Design models at various abstraction levels
Environment and tool dependency	Medium: • Need for a method base of method fragments	Low	High: • Need for transformation tool • Need for various method bases	High: • Need for transformation tool • Need for mapping method bases

- **Statelessness:** This refers to the preservation of the model state during the transformation; in other words, whether the source model is modified or kept unchanged during the transformation process.
- **Automatability:** Whether the transformation can be carried out automatically, or it can only be applied manually.
- **Understandability:** Whether the syntax of the transformation rules can be easily understood by the users.
- **Directionality:** Whether the transformation is bidirectional or unidirectional. Bidirectional transformations can be used in two different directions: to transform the source model(s) into target model(s), and conversely.
- **Evolvability:** Whether the transformation approach makes gradual updates possible.
- **Tracking:** Whether the transformation approach keeps track of the sequence of changes.
- **Tool support:** Whether the transformation approach provides tool support.
- **Syntactic correctness:** Whether a well-defined target model is producible from a well-defined source model.
- **Syntactic completeness (preservation of information):** Whether for each element of the source model, the related element is created in the target model by the model transformation.
- **Semantic correctness:** Whether the produced target model meets the conceptual features expected. This criterion can be a crucial requirement in transformations for which preservation of certain behavioral properties is important (such as refactoring transformations).
- **Termination:** Whether termination of the transformation process is guaranteed.
- **Confluence (Inconsistency):** Whether transformation results complement each other and are consistent.
- **Change propagation:** Whether changes in the source models can be automatically propagated to the target models.

The results of this evaluation are shown in Table 9. As seen in Table 9, most of the proposed criteria are satisfied by the transformation patterns. The strength of these patterns lies in the tool support provided for their automatic execution. Due to this feature, there is no need for the users to understand the QVT-equivalents of the transformation rules; therefore, low understandability (mentioned in Table 9) is not a significant weakness of the patterns, unless a user needs to edit the QVT code of a transformation pattern. Another weakness pointed out in the table is that termination of the transformation process cannot be guaranteed; however, this problem can be ameliorated if certain preconditions are defined and imposed by the method engineer.

5.4. Example of application of the proposed approach

The proposed approach has been applied to a real-world project as an example of its enactment, in order to demonstrate its applicability. The project was carried out at an Iranian software development company that specializes in developing Web-based, mobile and E-commerce applications. The aim was to develop a suitable methodology for a project targeted at developing a correspondence management system. In order to determine the requirements of the target methodology, the relevant situational factors (along with their range of possible values) were given to the stakeholders (Programmer, Technical Manager, Domain Expert, and Analyst) for evaluation. The factors and their values, as assigned by the stakeholders, are shown in Table 10 (the assigned values are underlined).

Table 9
Evaluation of transformation patterns.

Assessment Criterion (and possible values)	Evaluation result
Statelessness (Yes/No)	Yes: New models can be created without any changes to the source models.
Automatability (Yes/No)	Yes: Transformation patterns have been successfully implemented and used in the Medini-QVT automatic transformation tool.
Understandability (Yes/No)	No: Transformation rules are expressed formally in QVT, so they are not easily understandable for the user.
Directionality (Bidirectional, Unidirectional)	Bidirectional: Since it is possible to trace the changes made to the model to the transformation patterns applied, the proposed approach supports bidirectional transformation.
Evolvability (Yes/No)	Yes: The proposed approach supports gradual update.
Tracking (Yes/No)	Yes: The changes made to the source models are recordable.
Tool support (Yes/No)	Yes: Tool support has been provided by implementing the transformation patterns in the Medini-QVT tool.
Syntactic correctness (Yes/No)	Yes: By applying the transformation patterns to the source (requirements) model, a well-defined target (methodology) model is produced that is understandable to the user and possesses the key features expected from a software development methodology model.
Syntactic completeness (Yes/No)	Yes: If the source model satisfies the preconditions of the transformation patterns, for each element of the model, its related element(s) will be created in the target model.
Semantic correctness (Yes/No)	Yes: The target model is created through the use of transformation patterns that produce the key elements of software development methodologies according to the SPEM metamodel.
Termination (Yes/No)	No: If the transformation patterns are not implemented properly and the relationships between the source and target models are not well-defined, termination of the transformation process is not guaranteed.
Confluence (Yes/No)	Yes: The models resulting from the application of the transformation patterns are unique and consistent. Also, the execution of a transformation pattern does not interfere with other transformation patterns. On the other hand, applying the patterns at each level of the framework complements and completes the existing models. No inconsistency occurs in the methodology model in passing from one level to the next.
Change propagation (Yes/No)	Yes: A change in the source models is automatically propagated to the target models upon re-execution of the transformation patterns.

Table 10
Important situational factors of the example, and their assigned values (underlined).

Factor classification	Situational factor	Value
Environmental factors	Financial constraints on the project	No\Medium\High
	Variety of end users	Wide\Limited
	Schedule constraints on the project	Yes\No
	Importance of project in environment	Yes\No
Application domain factors	Required standardization level in development methodology	High\Low
	Importance of quality factors in development methodology	High\Medium
	System size	Large\Medium
	Criticality level	High\Medium
	Level of technology innovation required	High\Medium
	System's dependence on UI	High\Low
Project organization factors	Developers' business knowledge	Sufficient\Insufficient
	Developers' technical expertise	Sufficient\Insufficient
	Geographical distribution of development teams	Yes\No
	Distribution of skills among team members	Balanced\ Unbalanced
	Team's familiarity with agile methodologies	Sufficient\Insufficient

The requirements model of the example was created by applying the transformation patterns based on the values determined for each situational factor; the patterns used for this purpose were: "Identify the requirements corresponding to a given situational factor", "Identify parent-child relationships", "Identify dependency relationships among requirements", and "Identify influence relationships among requirements". Fig. 14 shows a partial requirements model for the example. In order to save space, the stakeholders involved have been intentionally left out of this figure, along with some of the less significant situational factors and their related requirements. A subset was then chosen from among the requirements (shown in Table 11).

As there are conflicts between some of these requirements, the conflict resolution algorithm is applied. As a result, some of these requirements are replaced. For example, "Welcome changes" and "Limit changes" are not compatible, and since "Limit changes" has a higher priority, the method engineer replaces "Welcome changes" with "Emphasize customer satisfaction"; the new requirement has been chosen from the requirements model, with the stakeholders' consent. Interviewing the project team determined that they had been using a customized version of Scrum for their projects. Also, the requirements determined in the previous step, such as "Limit modeling", "Speed up implementation", and "Emphasize customer satisfaction", showed the need for a lightweight methodology. Therefore, the scope (type) of the target methodology was set to "agile". The high-level phases and activities of the target agile methodology were determined by applying the "Extract general form of target methodology" transformation pattern; the resulting methodology model is shown in Fig. 15. This model and the requirements model were fed to the next level of transformation as input.

The "Map requirements to process patterns" and "Identify prerequisite process patterns" transformation patterns were then applied for extracting the method fragments corresponding to the requirements; the method fragments thus extracted are shown in Table 12. Finer-grained method fragments, consisting of roles and work products, were extracted from the R2MF method base by using the "Add roles to tasks" and "Add input and output products to tasks" transformation patterns. Inter-task relationships were then specified by using the "Determine transition relationships", "Define parallel relationships", and "Define synchronization relationships" patterns. This information was added to the methodology model and the Technique-Independent Methodology Model was created as a result. This model was fed to the next level of transformation

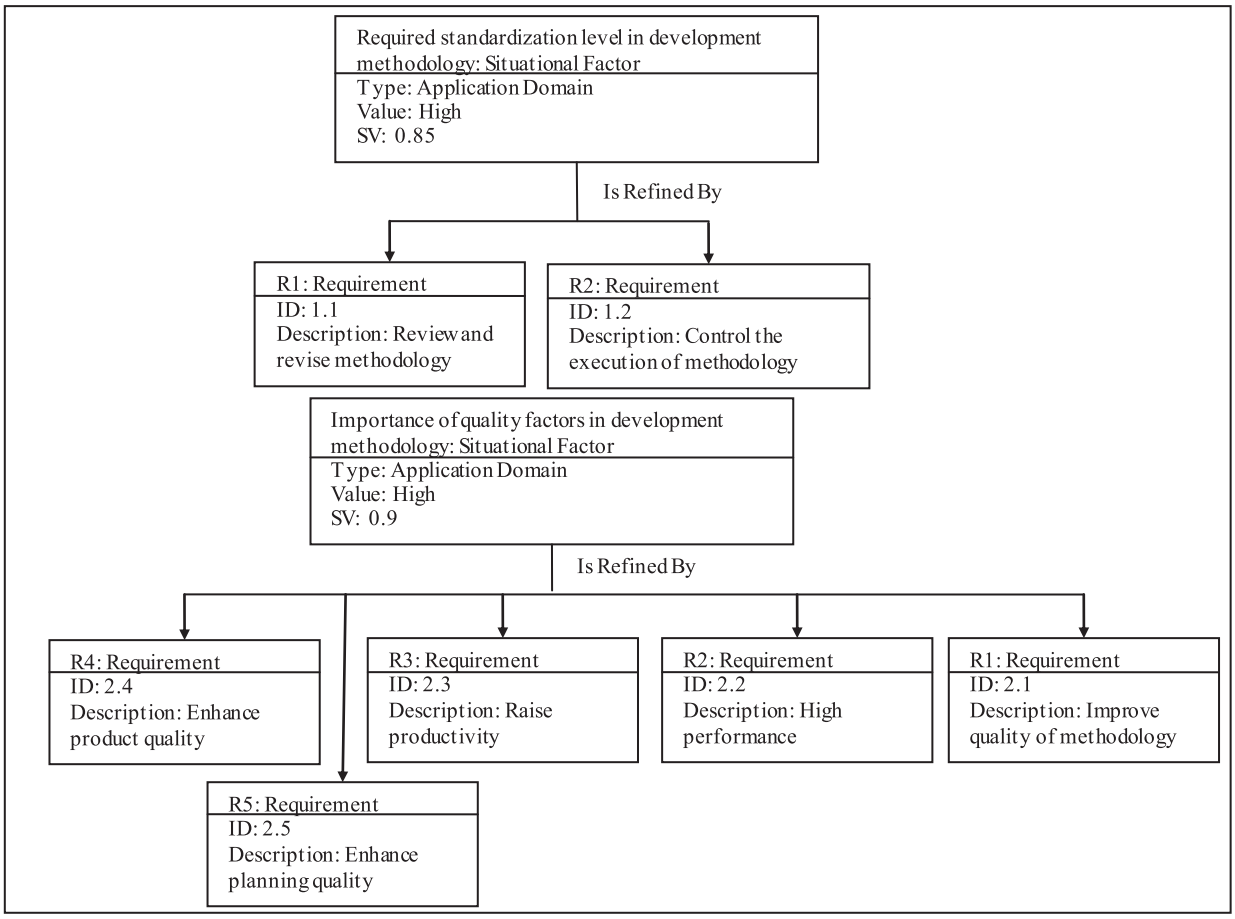

Fig. 14. Partial requirements model for the example.

Table 11
Requirements selected for the situational factors of the example.

Situational factor	Selected requirements
Schedule constraints on the project	• Obtain fast hardware • Prioritize requirements • Speed up implementation • Apply exact timing • Limit changes • Limit modeling
Importance of project in environment	• Use past project experiences • Apply evolution (maintenance)
Importance of quality factors in development methodology	• Improve quality of methodology • Raise productivity of people • Enhance planning quality
Required standardization level in development methodology	• Review and revise methodology
Criticality level	• Extract necessary models at each step • Obtain accurate hardware
System's dependence on UI	• Welcome changes • Apply UI-based software development • Enhance usability
Developers' business knowledge	• Perform domain analysis in each iteration • Improve business knowledge of team members
Developers' technical expertise	• Improve product quality • Improve skills of team members
Team's familiarity with agile methodologies	• Provide training on methodology

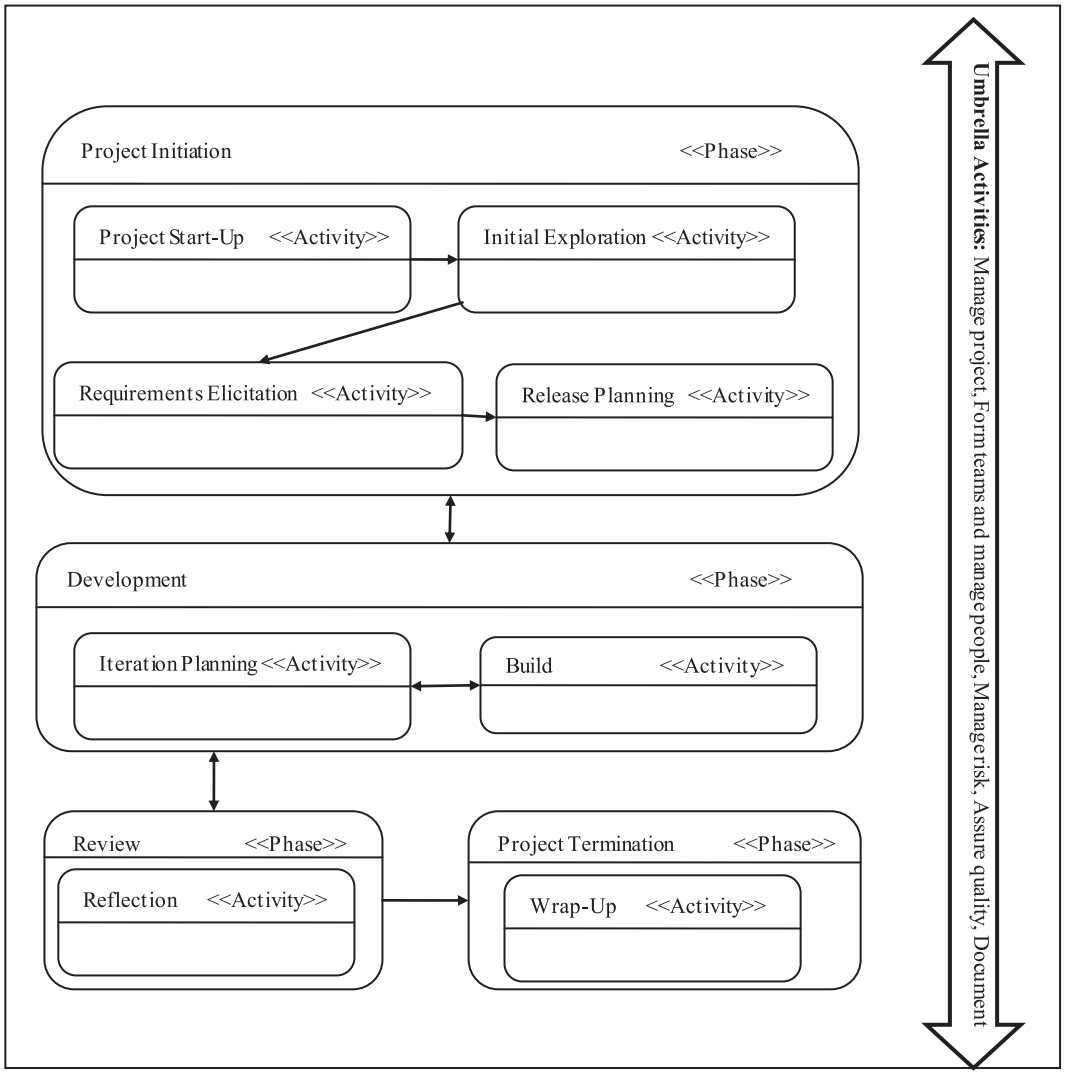

Fig. 15. High level phases and activities of the target methodology.

Table 12
Method fragments extracted for the requirements of the example.

Requirement	Extracted Method Fragments
Obtain fast hardware, Prioritize requirements, Speed up implementation, Apply exact timing, Limit changes, Limit modeling	Provide Software/Hardware Infrastructure, Prioritize Product Backlog, Code, Develop Rough Estimates, Declare Velocity, Evaluate Velocity
Use past project experiences, Apply evolution (maintenance)	Tune Development Teams, Refactor, Test, Code, Document Requirements, Generate Product Backlog, Design Architecture, Hold Design Sessions, Conduct Post-Mortem Activities
Improve quality of methodology, Raise productivity of people, Enhance planning quality	Recalibrate Release Plan, Tune Development Teams, Fine-Tune Methodology, (Re)Assign Responsibilities, Shape Methodology, Develop Release Plan
Review and revise methodology	Fine-Tune Methodology
Extract necessary models at each step, Obtain accurate hardware	Break Down into Tasks, Generate Product Backlog, Hold Design Sessions, Prepare Pilot Scheme, Design Architecture
Welcome changes, Apply UI-based software development, Enhance usability	Validate Product, Elicit User Experience, Build User Interface
Perform domain analysis in each iteration, Improve business knowledge of team members	Conduct Domain Walkthrough, Generate Product Backlog, Prioritize Product Backlog
Improve product quality, Improve skills of team members	Code
Provide training on methodology	Provide Training

as input. As the last step, the “Add techniques” pattern was applied for extracting the techniques related to each task. Fig. 16 shows the result of applying these transformation patterns (applied at the second and third levels of the framework); due to space limitations, only one of the activities (Requirements Elicitation) has been shown, along with its constituent method fragments.

The final model of the target methodology is shown in Fig. 17. In order to save space, some method fragments (such as roles, techniques and work products) have been intentionally left out. The methodology produced was ultimately demonstrated to the stakeholders. As the stakeholders had been using a customized version of Scrum for their projects, several suggestions were made

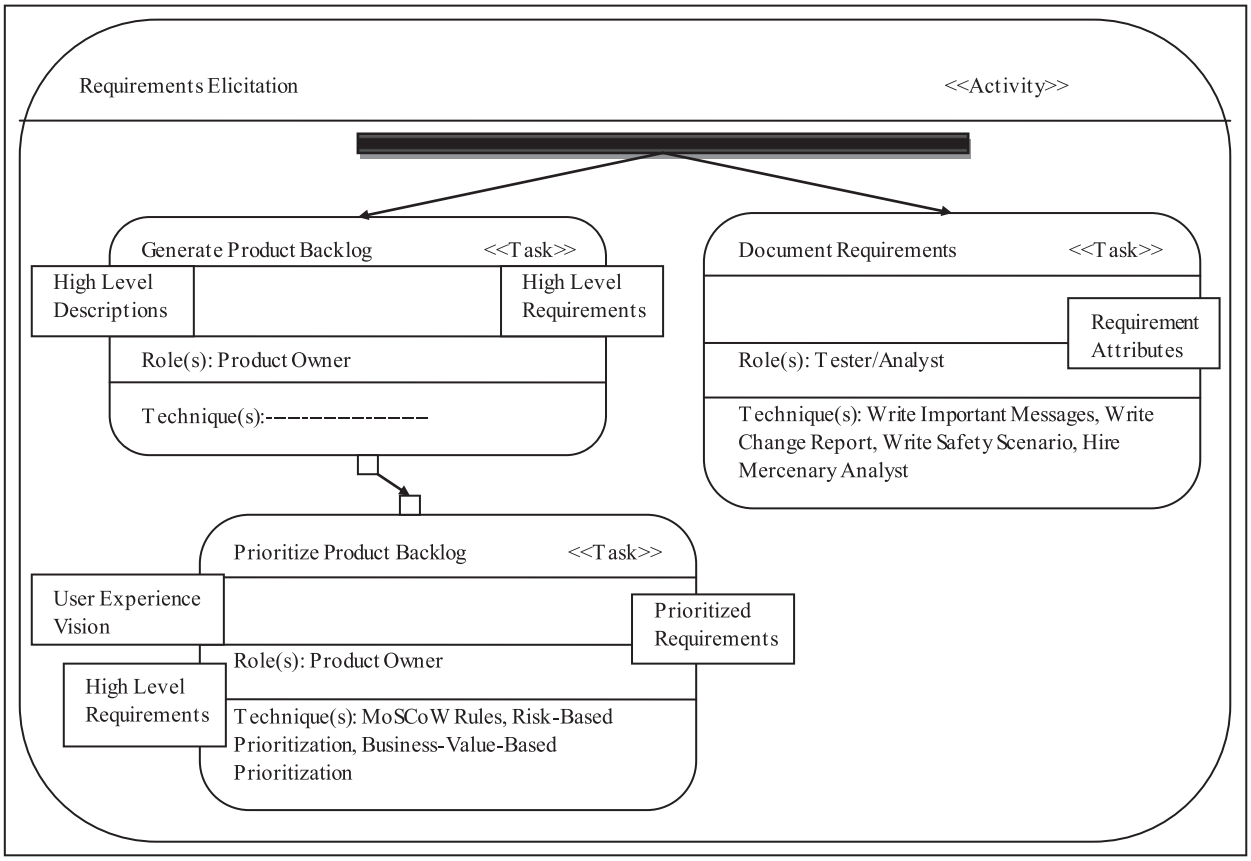

Fig. 16. Final model of the "Requirements Elicitation" activity.

for improving their current methodology based on the situational methodology produced.

The problems of the current methodology (as stated by the stakeholders) are tabulated in Table 13, along with the suggestions made for resolving them. Since the suggestions are based on the tasks of the situational methodology, explaining the suggestions requires a closer look at the tasks. The tasks based on which the suggestions were made are briefly explained below, along with their related suggestions:

1. **Shape methodology**: In this task, the skeleton of the methodology is shaped at the start of the process, to be revised at the end of each execution of the development phase. This task was not performed in the methodology practiced at the company. We therefore suggested that the methodology be augmented with this task, and that the methodology be evolved by using the "Hold Reflection Workshop" technique.
2. **Provide training:** Organizations that provide training on the use of the methodology are better prepared to implement it [58]. This task was not performed in the methodology practiced at the company. We suggested that training be conducted during the initial phases of the methodology, and that "Direct Training" and "Expert in Earshot" techniques be used for this purpose.
3. **Elicit user experience:** This task is essential for obtaining a thorough understanding of the user environment and the relevant requirements. It was partially covered by the methodology practiced at the company (in an implicit manner). We suggested that the current methodology be augmented with this task to enhance interaction with the users and identify their exact expectations from the system under development.
4. **Conduct domain walkthrough**: In this task, a high-level description of the problem domain is presented by the domain experts. This task was partially covered by the methodology practiced at the company during architectural design. We suggested that the "Conduct Domain Walkthrough" task be performed in the early phases of the methodology to enhance the organizational knowledge of the development teams. The "Hold Problem Domain Training Sessions" technique was proposed for conducting the task.
5. **Document requirements**: The information on the requirements, which is mostly exchanged through face-to-face communication, is documented through this task. This task was partially covered by the methodology practiced at the company; descriptions of the requirements were prepared by the Product Owner, but parts of the information that had been elicited through face-to-face communication were not documented. We suggested that the current methodology be augmented with this task, and that the "Discover Safety Scenarios", "Write Change Report" and "Hire Mercenary Analyst" techniques be used for conducting the task.
6. **Build user interface:** In this task, prototypes of the user interface are built and user feedback is obtained for improving the UI. This task was not performed in the methodology practiced at the company. We suggested that the methodology be augmented with this task to thereby increase user satisfaction.
7. **Develop release plan:** In this task, a list of requirements and goals with the highest priority are specified by the customer to be achieved in the next execution of the development phase. This task was partially covered in the methodology practiced at the company; however, it was only performed during the pregame phase of the methodology. We suggested that the task be added to the methodology in order to enhance the qual-

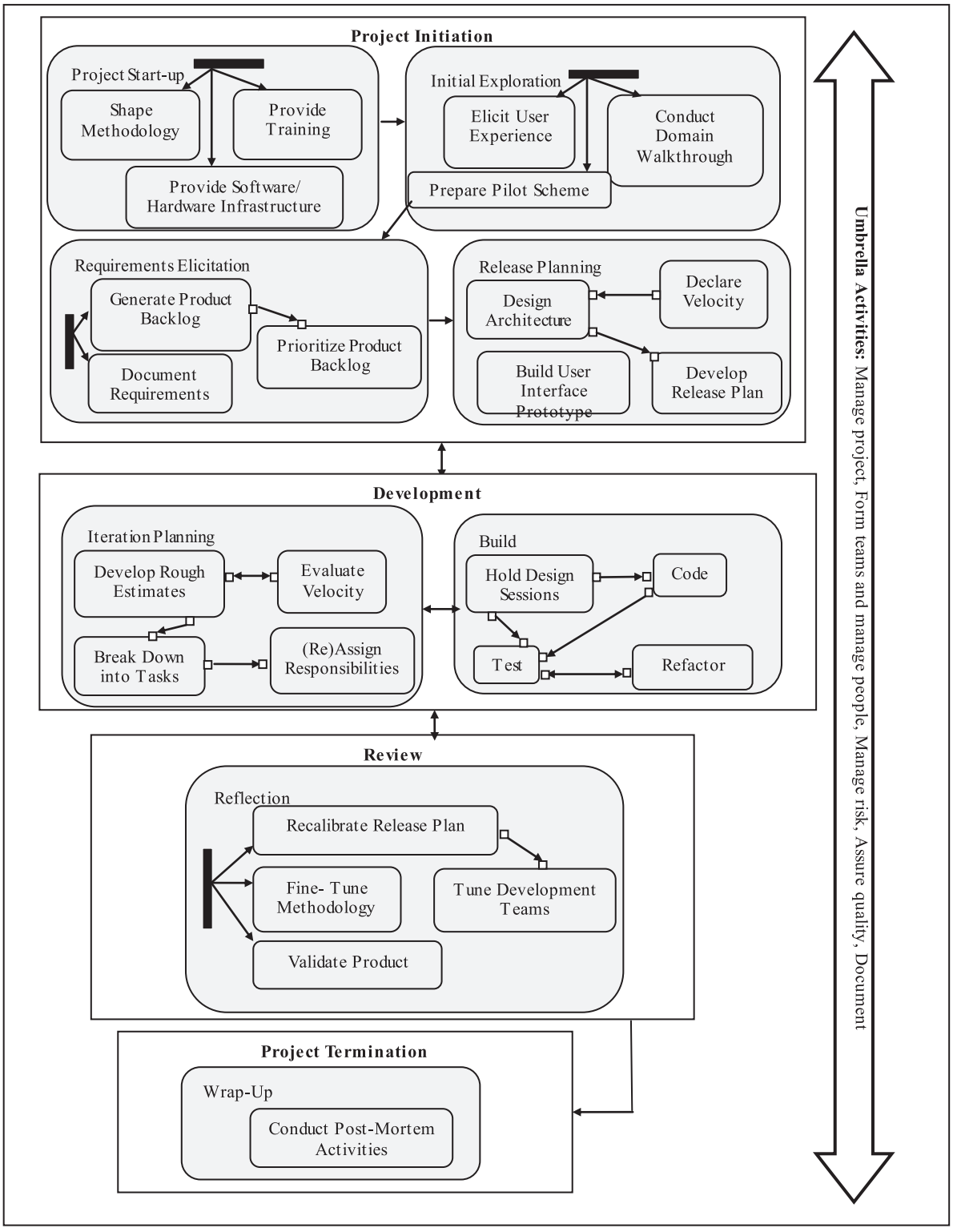

Fig. 17. Final model of the methodology produced in the example.

ity of the plans. Furthermore, we suggested the addition of the "Recalibrate Release Plan" task to the methodology, so that the plan is iteratively updated and refined.

8. **Code:** This task was completely covered by the methodology practiced at the company, but proper techniques were not used. We suggested that "Pair Programming" and "Side-by-Side Programming" techniques be applied in order to enhance software quality and facilitate skill transfer.
9. **Test:** This task was completely covered by the methodology practiced at the company, but proper techniques were not used. We suggested that the "Test-Driven Development" technique be applied in order to enhance software quality and increase customer satisfaction.

Table 13
Problems with the current methodology, and suggestions based on the situational methodology developed in the example project.

	Problem	Suggestion	Effectiveness of the suggestion			
			Ineffective	Low	Medium	High
1	Single, fixed methodology is used for all projects.	Evolve the methodology by using the "Hold Reflection Workshop" technique.			✓	
2	Methodology training is cumbersome, as it is performed during the project.	Conduct training by using "Direct Training" and "Expert in Earshot" techniques.		✓		
3	The Product Owner prepares the requirements as user stories. If he/she does not have the required knowledge for the task, the development team has to create user stories from textual files.	Enhance interaction with stakeholders by applying the "Elicit User Experience" task.				✓
4	Analysis of the problem domain is unwieldy, as it is performed through the "Design Architecture" task.	Enhance the organizational knowledge of the development team by using the "Hold Problem Domain Training Sessions" technique.				✓
5	Descriptions of the requirements are prepared by the Product Owner, but parts of the information, which has been elicited through face to face communication, is not documented.	Apply the "Document Requirements" task in the methodology, and use the "Discover Safety Scenarios", "Write Change Report" and "Hire Mercenary Analyst" techniques for performing the task.				✓
6	UI design is not specifically targeted in the current methodology.	Increase customer satisfaction by applying the "Build User Interface" task in the methodology.				✓
7	Project scheduling is only performed during the pregame phase of the current methodology.	Enhance the quality of plans by applying the "Develop Release Plan" and "Recalibrate Release Plan" tasks in the methodology.			✓	
8	Proper coding techniques are not applied.	Enhance software quality and facilitate skill transfer by using the "Pair Programming" and "Side-by-Side Programming" techniques.				✓
9	Proper software testing techniques are not applied.	Enhance software quality and increase customer satisfaction by using the "Test-Driven Development" technique.				✓

The "Effectiveness of the Suggestion" column in Table 13 shows the stakeholders' responses to the suggested improvements; it can be observed that they have shown a positive response to most of the suggestions.

6. Conclusions and future work

Evaluation results indicate that the proposed approach addresses the deficiencies encountered in previous approaches (especially as to automation and provision of method bases) by designing two method bases and a set of executable transformation patterns (13 transformation patterns), and implementing them in a specialized model transformation tool (Medini-QVT). On the other hand, the proposed approach is highly flexible as it allows the method engineer to actively join in and refine the produced methodology. The targeted quality attributes are improved as follows:

- Complexity is managed in three ways: 1- through the three-level modeling framework, and the modeling language applied; 2- by providing a semi-automated process in which most of the SME process tasks are performed automatically (the method engineer is only involved for further refinement); and 3- via breaking the SME process into fine-grained activities through which the target methodology is gradually completed.
- Portability is enhanced by defining a three-level modeling framework in which higher-level models are portable across situations and techniques.
- Productivity: Deciding about whether a specific task should be part of a methodology or not is very time-consuming; this problem is addressed in our approach through the provision of method bases and transformation patterns, so that suitable method fragments for the project at hand are extracted automatically through the execution of transformation patterns. Another important issue that can adversely affect productivity is the need for highly skilled professionals for software process definition, as this task requires experience and knowledge from several disciplines of software engineering [59]; this issue too is dealt with through the use of method bases and transformation patterns, which enable non-experts to perform the task. An important issue in productivity is to make sure that the quality of the target methodology is not traded for development speed; quality is properly maintained in our proposed approach, as the method fragments that are stored in the method bases have been extracted from existing methodologies, and have therefore been tried and tested in practice.
- Reusability: Reusability is enhanced in two ways: 1- software process knowledge is reused through the creation and use of method bases; and 2- the three-level MDD framework enhances the reusability of models: higher-level models are independent from situations/techniques, and are hence more readily reusable due to their relative abstractness.

We have demonstrated the applicability of the proposed approach by applying it to a real-world project. The method bases are only partially populated, but since the scope of the methodology was set to "agile" for this project, the contents were sufficient

for achieving the objectives. However, if the proposed approach is to be applied to other types of methodologies (scopes) that are not currently supported by the method bases, the content necessary for those types should be identified and added to the method bases; once the content has been added, it can be used for developing all the methodologies of the new types.

This research can be furthered in several directions: Completing the SF2R and R2MF method bases, refining the transformation patterns and extending the pattern set, and extending the applied tool with advanced SME features. Although the results of applying the approach to an industrial SME effort were encouraging, there still exist certain threats to their validity: the approach has been applied to a relatively small-scale SME project, and its applicability has not been fully explored in cases where the project situation is of a more complex nature; applying the approach to larger-scale SME projects has therefore been planned as a future undertaking, and is expected to improve the proposed approach and enhance the transformation patterns and method bases. We also acknowledge the merits of empirical experimentation for mitigating these threats, and plan to conduct such a validation in the future.

References

[1] B. Henderson-Sellers, J. Ralyté, P.J. Ågerfalk, M. Rossi, Situational Method Engineering, Springer-Verlag, 2014, doi:10.1007/978-3-642-41467-1.

[2] J. Ralyté, R. Deneckère, C. Rolland, Towards a generic model for situational method engineering, in: Proceedings of the Advanced Information Systems Engineering, 2003, pp. 95–110, doi:10.1007/3-540-45017-3_9.

[3] I. Mirbel, J. Ralyté, Situational method engineering: Combining assembly-based and roadmap-driven approaches, Requir. Eng. 11 (1) (2006) 58–78, doi:10.1007/s00766-005-0019-0.

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

[5] Z. Zohrevand, Y.M. Bibalan, R. Ramsin, Towards a framework for the application of model-driven development in situational method engineering, in: Proceedings of the 18th Asia Pacific Software Engineering Conference (APSEC'11), 2011, pp. 122–129, doi:10.1109/APSEC.2011.55.

[6] M. Völter, T. Stahl, J. Bettin, A. Haase, S. Helsen, Model-Driven Software Development: Technology, Engineering, Management, John Wiley & Sons, 2013.

[7] J. Mukerji, J. Miller, MDA Guide Version 1.0.1, Object Management Group, 2003 http://www.omg.org.

[8] T. Mens, K. Czarnecki, P. Van Gorp, A taxonomy of model transformation, in: Proceedings of the International Workshop on Group and Model Transformation (GraMoT'05), 2006, pp. 7–23, doi:10.1016/j.entcs.2005.10.021.

[9] Medini QVT: IKV++ technologies home, http://www.ikv.de.

[10] M.E. Iacob, M.W. Steen, L. Heerink, Reusable model transformation patterns, in: Proceedings of the 12th Enterprise Distributed Object Computing Conference Workshops, 2008, pp. 1–10, doi:10.1109/EDOCW.2008.51.

[11] I. Graham, B. Henderson-Sellers, H. Younessi, The OPEN Process Specification, Addison-Wesley, 1997.

[12] D.G. Firesmith, B. Henderson-Sellers, The OPEN Process Framework: An Introduction, Addison-Wesley, 2002.

[13] ISO/IEC. "ISO/IEC 24744:2007/Amd 1:2010 notation. Software Engineering – Metamodel for Development Methodologies," ISO, Geneva, 2010.

[14] ISO/IEC. "ISO/IEC 24744. Software Engineering – Metamodel for Development Methodologies," ISO, Geneva, 2007.

[15] F. Karlsson, P.J. Ågerfalk, Method configuration: adapting to situational characteristics while creating reusable assets, Inf. Softw. Technol. 46 (9) (2004) 619–633, doi:10.1016/j.infsof.2003.12.004.

[16] M. Cervera, M. Albert, V. Torres, V. Pelechano, A model-driven approach for the design and implementation of software development methods, Int. J. Inf. Syst. Model. Des. 3 (4) (2012) 86–103, doi:10.4018/jismd.2012100105.

[17] E. Breton, J. Bézivin, Model driven process engineering, in: Proceedings of the 25th Annual International Computer Software and Applications Conference (COMPSAC'01), 2001, pp. 225–230, doi:10.1109/CMPSAC.2001.960620.

[18] J.A.H. Alegría, M.C. Bastarrica, A. Quispe, S.F. Ochoa, An MDE approach to software process tailoring, in: Proceedings of the 7th International Conference on Software and Systems Process, 2011, pp. 43–52, doi:10.1145/1987875.1987885.

[19] T. Punter, K. Lemmen, The MEMA-model: Towards a new approach for Method Engineering, Inf. Softw. Technol. 38 (4) (1996) 295–305, doi:10.1016/0950-5849(95)01087-4.

[20] D. Gupta, N. Prakash, Engineering methods from method requirements specifications, Require. Eng. 6 (3) (2001) 135–160, doi:10.1007/s007660170001.

[21] A. Braganca, R.J. Machado, Transformation patterns for multi-staged model driven software development, in: Proceedings of the 12th International Software Product Line Conference (SPLC'08), 2008, pp. 329–338, doi:10.1109/SPLC.2008.41.

[22] J. Bézivin, F. Jouault, J. Paliès, Towards model transformation design patterns, in: Proceedings of the First European Workshop on Model Transformations (EWMT'05), 2005, pp. 1–6.

[23] S. Brahe, B. Bordbar, A pattern-based approach to business process modeling and implementation in web services, in: Proceedings of the Service-Oriented Computing (ICSOC'07), Springer, 2007, pp. 166–177, doi:10.1007/978-3-540-75492-3_15.

[24] B.K. Appukuttan, T. Clark, S. Reddy, L. Tratt, R. Venkatesh, A pattern based model driven approach to model transformations, in: Proceedings of the Metamodelling for MDA, 2003, pp. 110–128.

[25] K. Duddy, A. Gerber, M. Lawley, K. Raymond, J. Steel, Model Transformation: a declarative, reusable patterns approach, in: Proceedings of the 17th IEEE International Enterprise Distributed Object Computing Conference, 2003, pp. 174–185, doi:10.1109/EDOC.2003.1233847.

[26] A. D'Ambrogio, A model transformation framework for the automated building of performance models from UML models, in: Proceedings of the 5th international workshop on Software and Performance, 2005, pp. 75–86, doi:10.1145/1071021.1071029.

[27] C.V. Chicote, B. Moros, A. Toval, REMM-Studio: An integrated model-driven environment for requirements specification, validation and formatting, Object Technol. 6 (9) (2007) 437–454.

[28] A. Niknafs, R. Ramsin, Computer-aided method engineering: An analysis of existing environments, in: Proceedings of the Advanced Information Systems Engineering, 2008, pp. 525–540, doi:10.1007/978-3-540-69534-9_39.

[29] P. Clarke, R.V. O'Connor, The situational factors that affect the software development process: Towards a comprehensive reference framework, Inf. Softw. Technol. 54 (5) (2012) 433–447, doi:10.1016/j.infsof.2011.12.003.

[30] B. Henderson-Sellers, J. Ralyté, Situational Method Engineering: State-of-the-Art Review, Universal Comput. Sci. 16 (3) (2010) 424–478, doi:10.1007/978-3-642-41467-1.

[31] E. Kornyshova, R. Deneckere, R. Salinesi, Method chunks selection by multicriteria techniques: an extension of the assembly-based approach, in: Proceedings of the Situational Method Engineering: Fundamentals and Experiences, 2007, pp. 64–78, doi:10.1007/978-0-387-73947-2_7.

[32] J. Jantzen, Foundations of Fuzzy Control, John Wiley & Sons, 2007, doi:10.1002/9780470061176.

[33] R.E. Precup, H. Hellendoorn, A survey on industrial applications of fuzzy control, Comput. Ind. 62 (3) (2011) 213–226, doi:10.1016/j.compind.2010.10.001.

[34] M. Sadiq, S.K. Jain, Applying fuzzy preference relation for requirements prioritization in goal oriented requirements elicitation process, Int. J. Syst. Assurance Eng. Manage. 5 (4) (2014) 711–723, doi:10.1007/s13198-014-0236-3.

[35] O. Jafarinezhad, R. Ramsin, Development of situational requirements engineering processes: a process factory approach, in: Proceedings of the 36th Annual IEEE Computer Software and Applications Conference (COMPSAC'12), 2012, pp. 279–288, doi:10.1109/COMPSAC.2012.39.

[36] A. Van Lamsweerde, R. Darimont, E. Letier, Managing conflicts in goal-driven requirements engineering, IEEE Trans. Softw. Eng. 24 (11) (1998) 908–926, doi:10.1109/32.730542.

[37] A. Sardinha, J. Araújo, A. Moreira, A. Rashid, Conflict Management in Aspect-Oriented Requirements Engineering, Inf. Sci. Technol. Bull. ACM Slovakia 2 (1) (2010) 56–59.

[38] H. Bendjenna, P.J. Charrel, N.E. Zarour, Using AHP method to resolve conflicts between non-functional concerns, in: Proceedings of the International Conference on Education, Applied Sciences and Management, 2012, pp. 167–170.

[39] M. Amroune, J.M. Inglebert, P.J. Charrel, N. Zarour, A Conflict Resolution Process in AspeCiS approach, Int. J. Comput. Appl. 44 (10) (2012) 14–21, doi:10.5120/6298-8504.

[40] K. Mitchell, B.R. Agle, D.J. Wood, Toward a theory of stakeholder identification and salience: defining the principle of who and what really counts, Acad. Manage. Rev. 22 (4) (1997) 853–886, doi:10.5465/AMR.1997.9711022105.

[41] S.W. Ambler, Process Patterns: Building Large-Scale Systems Using Object Technology, Cambridge University Press, 1998.

[42] R. Bendraou, M.P. Gervais, X. Blanc, Uml4spm: a uml2.0-based metamodel for software process modelling, in: Proceedings of the Model Driven Engineering Languages and Systems, Springer, 2005, pp. 17–38, doi:10.1007/11557432_3.

[43] R. Bendraou, J.M. Jézéquel, M.P. Gervais, X. Blanc, A comparison of six uml-based languages for software process modeling, IEEE Trans. Softw. Eng 36 (5) (2010) 662–675, doi:10.1109/TSE.2009.85.

[44] R. Schuppenies, S. Steinhauer, Software Process Engineering Metamodel, Object Management Group, 2002.

[45] S. Hesari, H. Mashayekhi, R. Ramsin, Towards a general framework for evaluating software development methodologies, in: Proceedings of the 34th Annual IEEE Computer Software and Applications Conference (COMPSAC'10), 2010, pp. 208–217, doi:10.1109/COMPSAC.2010.69.

[46] R. Bendraou, B. Combemale, X. Cregut, M. Gervais, Definition of an executable SPEM 2.0, in: Proceedings of the 14th Asia-Pacific Software Engineering Conference, 2007, pp. 390–397, doi:10.1109/ASPEC.2007.60.

[47] M. Asadi, R. Ramsin, Patterns of situational method engineering, in: Proceedings of the Software Engineering Research, Management and Applications, 2009, pp. 277–291, doi:10.1007/978-3-642-05441-9_24.

[48] Z. Shakeri, M.H. Sadi, R. Ramsin, Towards tool support for situational engineering of agile methodologies, in: Proceedings of the 17th Asia Pacific Software Engineering Conference (APSEC'10), 2010, pp. 326–335, doi:10.1109/APSEC.2010.45.

[49] M. Asadi, N. Esfahani, R. Ramsin, Process patterns for MDA-based software development, in: Proceedings of the 8th ACIS International Conference on Software Engineering Research, Management and Applications (SERA'10), 2010, pp. 190–197, doi:10.1109/SERA.2010.32.
[50] M. Khaari, R. Ramsin, Process patterns for aspect-oriented software development, in: Proceedings of the 17th IEEE International Conference and Workshops on Engineering of Computer Based Systems (ECBS'10), 2010, pp. 241–250, doi:10.1109/ECBS.2010.33.
[51] E. Kouroshfar, H.Y. Shahir, R. Ramsin, Process patterns for component-based software development, in: Proceedings of the Component-Based Software Engineering, 2009, pp. 54–68, doi:10.1007/978-3-642-02414-6_4.
[52] B. Biglari, R. Ramsin, Generic process framework for developing high-integrity software, in: Proceedings of the 11th International Conference on Intelligent Software Methodologies, Tools and Techniques (SoMeT'12), 2012, pp. 73–88, doi:10.3233/ 978-1-61499-125-0-73.
[53] R. Babanezhad, Y.M. Bibalan, R. Ramsin, Process patterns for web engineering, in: Proceedings of the 34th Annual IEEE Computer Software and Applications Conference (COMPSAC'10), 2010, pp. 477–486, doi:10.1109/COMPSAC.2010.55.
[54] M. Karow, A. Gehlert, J. Becker, W. Esswein, On the transition from computation independent to platform independent models, in: Proceedings of the 12th Americas Conference on Information Systems (AMCIS'06), 2006, pp. 3913–3921.
[55] A.A.A. Jilani, M. Usman, Z. Halim, Model transformations in model driven architecture, Universal J. Comput. Sci. Eng. Technol. 1 (1) (2010) 50–54.
[56] T. Yue, L.C. Briand, Y. Labiche, A systematic review of transformation approaches between user requirements and analysis models, Require. Eng. 16 (2) (2011) 75–99, doi:10.1007/s00766-010-0111-y.
[57] F. Orejas, E. Guerra, J. de Lara, H. Ehrig, Correctness, completeness and termination of pattern-based model-to-model transformation, in: *Proceedings of Algebra and Coalgebra in Computer Science*, 2009, pp. 383–397, doi:10.1007/978-3-642-03741-2_26.
[58] J.A. Livermore, Factors that significantly impact the implementation of an agile software development methodology, J. Softw. 3 (4) (2008) 31–36, doi:10.4304/jsw.3.4.31-36.
[59] A. Barreto, E. Duarte, A.R. Rocha, L. Murta, Supporting the definition of software processes at consulting organizations via software process lines, in: Proceedings of Quality of Information and Communications Technology (QUATIC), 2010, pp. 15–24, doi:10.1109/QUATIC.2010.19.