A Step towards Method Configuration From Situational Method Engineering

Daya Gupta and Rinky Dwivedi

Computer Engineering Department, Delhi Technological University

Bawana, New Delhi, INDIA

Daya_gupta2005@yahoo.co.in, rinkydwivedi@gmail.com

Abstract-**We define a two part SME process having a configuration part and an assembly part. The former is to obtain methods that are configurable and individually meet partial requirements of the desired method. The assembly process then puts these methods together to form a coherent situated method. Thus, the SME process relies on the configuration sub process to provide a degree of assurance that the required situated method shall indeed be produced. We show how configurable methods can be produced and how these can be configured. The point when assembly is to be done is identified. An outline of the method base to support the SME process is presented.**

Keywords-**Configuration process, Situational method engineering, Variant.**

1. Introduction

Situational method engineering (SME) is the technique for constructing Information System Development Methods that address specific project needs. It relies on a method base that is the repository of method components. These components can be retrieved and assembled together to form the desired method. A number of approaches exist for component retrieval. These rely on notions of descriptors [1], project contingency factors [2] or method contexts [3]. Components are defined in compliance with Meta models and may be fragments [4], contexts [5], decisions [6] etc. The assembly technique itself has been illustrated in [9] for putting together state chart and object model.

A number of issues arise in these approaches to SME. The first issue is that of the appropriateness of the components retrieved. The retrieved component may or may not be found suitable to form the desired method during the assembly process. The second issue is of ensuring coherence of an assembly of components. Evidently, to get a coherent method, it is not enough to assemble only coarse grained components but the assembly process must go down to the finest grained components. Since assembly is to be done for all levels of granularity, it becomes a detailed tedious task.

Recently there are proposals to provide a rich set of guidelines and structured approaches [8, 10] to form a coherent method. These proposals are analogous to software engineering domain and are two staged first architecture of situated method is formed and then method is organized from this architecture. Thus method construction task is performed in a more disciplined and cohesive way. Still the issues of appropriateness of the method component being selected remain unanswered.

In recent times, the SME has moved to method configuration to construct a project specific method [20, 21]. They rely on base method/method components which can be transformed into a situation specific method through process of tailoring, extension or assembly. Recently configurability is gaining importance in industrial domain, two important case studies that reflect the practical application of this approach Case study of IBM global services on configurability of Work Products [12] and Case Study of Intel Shannon configuring agile methods [13]. We now shall extent the notion of configurability to situation method engineering.

We draw an analogy between method configuration and system configuration. The system configuration is based on the construction of a 'configurable system model' that represents the essential system concepts and inter-relationship between these. The configuration process then considers these to form the new system. It is assumed that the configurable system model is sufficiently generic to be specialized into a range of systems.

A number of issues need to be addressed for method configuration. The first is of a Meta model to be used to model the concepts of configured method. The second issue is what 'good component' is and what the 'right granularity' is. The third is regarding the Selection of a method component and lastly the process of configuration to reach coherent desired method.

We approach the first issue by modifying the decisional Meta model of Prakash [6] to reach configurable Meta model capable enough to model the concepts of our configurable method. The proposed configurable method can be atomic, compound, transformational or constructional and it exhibits characteristic which can either be *common* or *variable*. Commonality in our view is the characteristic of a method which if configured, the method will lose its identity and Variability is the characteristic of a method which can be configured.

Secondly we define method component are entire methods, whether atomic or compound. From forgoing these method models are configurable and will have characteristics of commonality and variability. This is also the right granularity because such methods provide to us the most basic, coherent assembly of method components. Coming to the third issue, that of ensuring appropriateness of the retrieved component. As the method base now contains methods, we propose to do retrieval based on global properties of methods like stage of the life cycle for which the method is applicable and method concepts used. Since the retrieved components shall satisfy global method properties, the chance of retrieving relevant components becomes high.

Finally after retrieving the configurable component we configure them as per requirements. Now with configured method, any assembly with other components can be carried out. This reduces the assembly process to manageable proportions: there is some assurance that the components being assembled have a reasonable chance of producing the desired method and the detailed work of assembling components is limited to such components. Thus our proposal is to make a two stage SME process. The first stage is the configuration of retrieved methods and the second is to assemble the configured methods to form the situated method.

In the next section we review the related research and formulate our proposal for method configurability by analogy with system configurability. In section 3, we build a Meta model for method configurability by introducing notions of commonality and variability in the basic Meta model of [6]. In section 4 we apply the process of defining configurable methods to UML and represent it as a configurable model. Section 5 contains our configuration process. This process takes a configurable method as input and produces a configured method. We consider our two stage SME process in section 6. Here we consider the method base of configurable components, retrieval from it, and application of the configuration process. Finally, an identification of where assembly is needed. Section 7 contains a discussion of related work.

2. Method configuration

In this section we illustrate method configurability. We first review work on configurability, commonality and variability in systems development, situational method engineering and then present our view for method engineering. We will see that the configuration process concentrates on variable aspects of a method to yield the method required for a specific situation.

Configurability has been defined in many ways. The IEEE glossary considers it to be "The arrangement of a computer system or component, defined by the number, nature, and interconnections of its constituent parts". In the context of business process models, it has been treated as dealing with the question "How to model business processes that are similar to one another in many ways, yet differ in some other ways from one organization, project or industry to another?"[14].

The task of configurability is to first create a new model called a configurable model followed by selecting those parts of the configurable model that are relevant to the user's requirement. Configurable models use notions of commonality and variability. Coplien et al [15] define **commonality** as an assumption held uniformly across a given set of objects whereas **variability** is an assumption that is true for only some elements of the set. In [16] we have the definition of **variability** as "an assumption about how members of a family may differ from one another". A configurable model identifies commonality and variability that can be exploited in developing a new system from the configurable model.

Davenport [17] describes the process of configuration as a methodology performed to allow a business to balance their IT functionality with the requirements of their business. Soffer et al. [18] consider configuration as an alignment process of adapting the enterprise system to the needs of the enterprise. [19] Proposes a method that systematically develops requirements using commonality and variability in product line approaches.

Recently configurability is gaining importance in industrial domain, two important case studies that reflect the practical application of this approach Case study of IBM global services on configurability of Work Products

[12] and Case Study of Intel Shannon configuring agile methods [13].

Configurability based SME process relies on a Base Method. [20, 21] this is in contrast to earlier approaches for situation method engineering that relies on a method base. Formal definition of base method is given in [20] as "A base Method is the systems engineering method chosen as the starting point for the configuration work". The essential activity of configurability based SME approach is method tailoring, here the base or candidate method is tailored as per the situational need. The notion of method tailoring can be in any form whether reduction [22] or extension [22]. Karlsson [20] has presented a frame work for configuring methods called MMC (Method for Method Component). Here project specific method is configured from base method by administrating configuration packages. However, Wistrand [21] has defined a conceptual construct named as 'Method Component' which aims to transfer a base method for given goals in form of required artifacts. These proposals are in the infant stage.

We extend the notion of commonality and variability to method engineering. We assume that a configurable model of a method is an association of commonality and variability with method concepts. Table 1 states that ER has method components that can be qualified as common and variable, a method derived from ER must have the notions of *entity* and *relationship sets, attributes, primary key, role* and *cardinality*. However, notions of *weak entity set, arity of relationships* and *multiplicity of attributes* are variable.

TABLE 1
A CONFIGURABLE MODEL OF ER

Method concepts	Essentiality
Entity set	Common
Relationship set	Common
Weak Entity set	Variable
Attribute	Common
Primary Key	Common
Role	Common
Cardinality	Common
N-ary relationship	Variable
Multiplicity of attribute	Variable

By analogy with the process of configuration [18], we define the method configuration process as an alignment of a method to the needs of the situated method. Now just as the system configuration process yields a configured system belonging to the family so also the method configuration process produces a family of methods. A few members of the family of Table 1 are shown in Table 2. The configuration process includes all common concepts of ER in a method family member but selects variable ones based on the need of the specific project being handled. Table 2 shows three ER family members obtained through three instances of the configuration process.

TABLE 2
SOME FAMILY MEMBERS OF ER

Family Member	Concepts
1	No weak entity set, all others as in ER
2	All ER concepts but only single valued attributes
3	All ER concepts but binary relationships only

3. A META MODEL FOR METHOD CONFIGURABILITY

In this section we develop a Meta model for configurable method models. The decisional Meta model of [6] has a generic model part that treats a method as a triple < MB, Dep, E> where MB is a set of method blocks, Dep is a set of dependencies between these, and E is the enactment algorithm. Our generic configurability model introduces commonality and variability concepts in this basic generic model as shown in Fig. 1. The generic configurability model is centered round MB and Dep. E is the procedure that exploits the given set of MB and Dep to produce the product. It cannot be configured but comes as a given with the Meta model. The set Dep establishes dependencies between instances of method blocks. Thus, if a method block is common then all dependencies in which it participates are relevant to the configured method. However, if a method block is a variant and not included in the configured method then all dependencies in which these variants participate are meaningless. Since Dep is configured by the very act of inclusion/exclusion of method blocks, it is not to be directly configured by the method engineer. We treat it as not configurable in our Meta model.

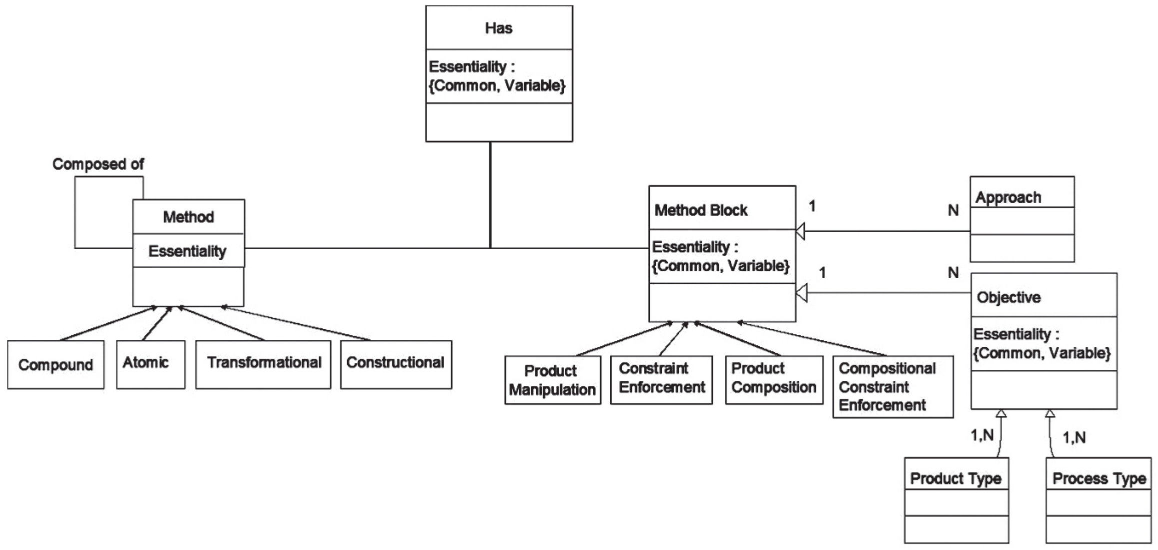

FIG. 1: THE CONFIGURABLE META MODEL

Fig. 1 shows the presence of an attribute called essentiality. Essentiality=common specifies commonality whereas Essentiality=variable specifies a variant. This figure shows that a method can be common or variable. This has particular relevance for compound configurable methods for example, UML which is compound method consisting is a unification of atomic methods:

< Use Case Diagram, Activity Diagram, Class Diagram, Sequence Diagram, Collaboration Diagram, State Chart Diagram, Deployment Diagram, Component Diagram>.

It is possible to declare Use Case Diagram [UCD] and Class Diagram as common and other component as variable. Any method configured from this shall necessarily have a Use Case Diagram and Class Diagram components whereas the others are optional. In contrast, an atomic configurable method can only be common.

Within a method, it is possible for method blocks to be either common or variable. This is shown in the Fig. 1 by the essentiality attribute of the concept method block. Thus in the foregoing example, the common class diagram can have its individual concepts as common or variable. For example, we may define an *object class* as common but an *operation* of the class as variable. Similarly in UCD we can define *actor, use case, communication* as essential whereas *generalization* can be variable.

Our basic process of defining configurable methods is as follows:

- Define the scope of the configurable method by identifying the family members.
- If the method is compound then define the essentiality property of each component method else define its essentiality as common.
- For every concept of the method, define the essentiality property.

This process is top down in the sense that we first establish essentiality for the global method and then proceed down to determine essentiality of coarse grained concepts and eventually to the finest grained ones.

As mentioned earlier, the decisional Meta model is an instantiation of the generic model. Correspondingly the configurable decisional method model is an instantiation of the generic configurable model. The instantiation relevant to our purposes is shown in Table 3.

TABLE 3
THE DECISIONAL META MODEL

Generic Model Concept	Decisional Meta Model Concept
Method block	Decision
Objective	Purpose
Product type	Structure
Process type	Operation

A method block is an aggregate of approach and purpose. For simplicity, let us ignore the notion of an approach. Thus a method block reduces to a purpose. Now, in a purpose, there is a structure part and an operation part. As we will see, the set of operations are a given in the Meta model. Thus, they are not configurable. The only configurability is of the concept structure. This results in the purpose and consequently, the decision to be configurable. Again, however, this configurability can be algorithmically determined, only that subset of purposes/ decisions is included in the situated method which is built on the included concept structures. Thus, there is no need for the method engineer to explicitly do this configuration.

In the rest of this section, we consider configurable structures and outline the set of non-configurable operations of the configurable decision Meta model.

3.1 Structure

There are two kinds of structures, those whose instances can be created and destroyed by application engineers and those whose instances are pre-defined. The former are called conceptual structures and the latter are called fixed structures. Conceptual structures constitute the set of concepts in terms of which a product is expressed. Fixed structures are those that are defined once and all for by a method engineer. An example of fixed structure is a method constraint such as completeness and conformity which cannot be created or destroyed by the application engineer.

Conceptual Structures

As shown in Fig. 2, conceptual structures are partitioned into two dimensions. The first dimension classifies them as either simple or compound. The second dimension represents conceptual structures into disjoint classes of structures called constraint, definitional, constructional, link, and collection of concepts respectively.

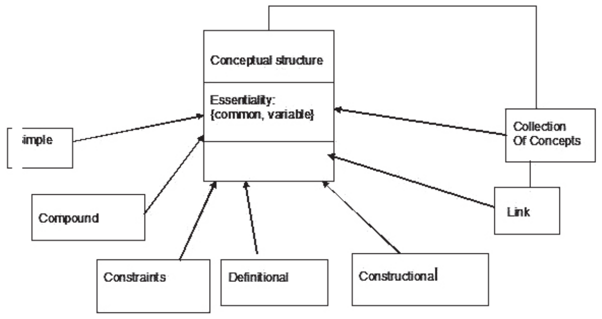

FIG. 2: THE CONFIGURABLE CONCEPTUAL STRUCTURE

Simple *constructional* structures cannot be decomposed into other components. *Links* are conceptual structures that are used to build collections of concepts from given concepts. For example ISA and aggregation are links, as they build abstraction hierarchies. *Collections of concepts* are constructed whenever constructional structures are connected by links. Aggregations, specialization hierarchies, and subtype hierarchies are examples of collection of concepts. A collection of concept is complex if it is defined out of other collections. *Definitional* structures define the properties of conceptual structures.

Constraints impose application-related constraints on conceptual structures. For example, such a constraint could say that the ages of employees should be less than 65 years.

The presence of the attribute, essentiality, in Fig. 2 shows that conceptual structures are configurable. We will illustrate this configurability in section 4 with an example.

Fixed Structures

Fixed structures deal with the restrictions that are used to enforce quality of conceptual structures. They are defined by the method engineer to help the application engineer in creating well defined and well formed conceptual structures. In their simplest form, they are the method constraints of completeness, consistency, conformity and fidelity.

Similarly there are compositional constraints which are specified between conceptual structures of the different simple component methods of a compound method. A structure of one of these cannot compose any arbitrary structure of the other. Such composition is governed by constraints that control the product resulting from the use of compound methods. The method engineer defines these constraints at the time the compound method is defined. For example in UML, *operation* in Class Diagram must be a use case in Use Case Diagram.

Fixed structures are shown in Fig. 3. Notice that they are configurable due to the presence of the essentiality attribute.

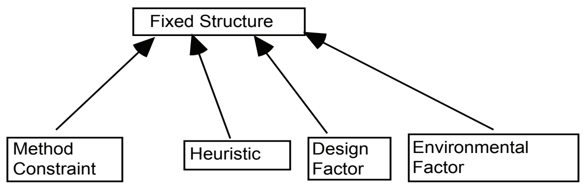

FIG. 3: CONFIGURABLE FIXED STRUCTURES

3.2 The Operation

Operations identify the set of process types that operate upon product types to provide product manipulation and verification capability to application engineers. Operations are classified into two four classes as follows:

- *Basic Life Cycle:* For each conceptual structure there are operations to *create,* and *delete* it. Create and delete are not defined for the fixed structures.

- *Relational:* These allow different structures to be related to one another. *These are attach, join,* couple, *associate,* relate*, apply* and their inverses.
- *Integration* This class of operations is defined for compound methods. These operations are, *export*, *import*, *correspond*, *convert* and their inverse operations.
- *Constraint Enforcement:* For each conceptual structure of a method and the method constraint applicable to it, a *method constraint enforcement* operation is defined.

4. UML AS A CONFIGURABLE METHOD

We apply the process of constructing a method described in section 3 to Unified Modeling Language which is composed of nine component methods such as Class Diagram, Sequence Diagram, Collaboration Diagram, State Chart Diagram, Component Diagram etc. We are assuming following qualification for these components< common, variable> as given below

TABLE 4
ESSENTIALITY OF UML METHOD COMPONENTS

Method component	Essentiality
Use Case Diagram	Variable
Activity Diagram	Variable
Class Diagram	Common
Sequence Diagram	Variable
Collaboration Diagram	Variable
State Chart Diagram	Variable
Component Diagram	Variable
Deployment Diagram	Variable
Object Diagram	Common

This table shows that the Class Diagram and Object Diagram are considered as essential to any method configured from UML. This means that UML can yield a family of methods that may be object oriented or data oriented [23] in nature. The assumptions about commonality and variability for each method concept of Class Diagram are shown in Table 5.

TABLE 5
CLASS DIAGRAM AS A CONFIGURABLE MODEL

Method concepts	Essentiality
Class	Common
Data_type	Common
Association	Common
Aggregation	Variable
Operation	Variable
Generalization	Variable
Generalization-link	Variable
Aggregation-link	Variable
Cardinality	Common
Degree of association	Variable
Multiplicity	Variable
Degree of association	Variable

The Class Diagram configurable model has *class, Data_ type, association and cardinality* as common to its family. The rest of the Class Diagram concepts are variants.

5. THE CONFIGURATION PROCESS

The configuration process assumes the existence of a configurable method. In producing a method, the process first considers the essentiality property of the global method. If the method is atomic then it is accepted as such and renamed. If it is a compound method then all components that have Essentiality=common are accepted in the method To-Be. Similarly, all method concepts of selected methods having Essentiality=common are accepted in the new method.

The focus of the configuration process now shifts to variants. These are to be examined by the method engineer for appropriateness for the new method. If relevant then they are included else they are excluded. Thus, the intellectual task of the method engineer is centered on variants.

It may happen that the new method needs concepts that are not present in the configuration produced. In such a case we propose to do method assembly by looking at other methods that may have these concepts.

As an example, let us configure Class Diagram to yield the ER model. As mentioned above, Class diagram has Essentiality = common. It is therefore renamed as ER and is now configured: all common concepts are present in ER (with *entity set* as alias for *class* and *relationship* set as alias for *association*). Out of the variants only *multiplicity* of *attributes* is taken. However, some concepts of ER, like *primary* key are not configurable from Class Diagram.

6. THE SME PROCESS

Our SME process consists of two sub processes, the configuration sub process and the assembly sub process. We propose that the configuration process is basic to the SME process and that assembly should be attempted only when configurability fails to deliver the desired method. This failure can happen in two ways:

- An entire method of a compound method is missing in the configurable method.
- Method concepts are missing.

Our SME process relies on the existence of a method base of configurable methods. Configurable methods can be retrieved based on global properties. These properties provide a broad indication of the family of methods that can be produced. We propose properties as follows:

(i) The part of the systems development life cycle catered to, requirements engineering, system design, complete life cycle etc.
(ii) The nature of the configurable method. This can be atomic or compound. These methods are further classified as proposed by [23] as data, process, or behavior oriented. Additionally, methods can be object oriented. Requirements engineering methods can be model driven or not, goal oriented, scenario oriented, and so on.
(iii) The application area of the method, whether constructional or transformational [24].
(iv) Component methods of compound methods having essentiality=common.

Additionally, for each method in the base, the method concepts that have essentiality = common can be used to retrieve these concepts.

This yields a method base structure (only relevant part shown. Details can be obtained from the authors.) Consisting of relations as follows:

I. Method_base(Method_id, method_name, life_cycle, application, method_type, method_nature). This relation keeps global method information: life_cycle identifies the stage of the life cycle addressed; application takes on values from {constructional, transformational}; method_type tells us whether the method is atomic or compound; method_nature is the method class like data oriented, process oriented.
II. Compound_Method(method_id, component_id, essentiality). This relation keeps track of the component methods of a compound method.
III. Configurable method (method_id, method _concept, essentiality). Commonality and variablility information about the concepts of a method are maintained.

Retrieval from the method base starts off by first retrieving configurable methods of interest. For example,

Select method_id, method_name, method_type
From method_base
Where life_cycle = system design application = constructional
And method_nature = data- oriented.

Evidently, the intention is to build a data oriented method for constructing system designs.

Individual methods from the list obtained can then be examined. Assume that a compound method is retrieved. Now, the method engineer obtains its component methods along with their essentiality in the compound method. For example, if a method named UML is having method_id=27 had been retrieved then the following query is made

Select component_ id, essentiality
From Compound_Method
Where method _id = 27

At this moment the method engineer knows which component methods are common and which are variants. The method engineer can now decide about the appropriateness of the configurable method for the situated method.

Once the configurable method is selected based on global properties, a detailed examination at the method concept level is performed. For example, if a component id=35, say Activity Diagram, is a potential candidate for inclusion in the situated method then a query to examine it can be formulated:

Select method concept
From configurable _method
Where method_ id = 35 and essentiality = common

As a result, all the retrieved common method concepts are examined for their appropriateness for the situated method. If these are acceptable then the variants can be obtained and their suitability assessed.

A list of missing methods/concepts is now made. The method base is searched again for methods that may contain these concepts. Thus a collection of configured methods is obtained that need to be assembled together to form the situated method.

To sum up, three cases arise:

1. A configurable method retrieved from the method base cannot be configured to yield the situated method. It is discarded and another one is considered.
2. The configured method is the method To-Be. In this case no assembly is required.

3. The configured method only partially meets the requirements of the method To-Be. Evidently, assembly with other configured methods is now to be done.

7. Related work

A body of knowledge exists for configuring information/ software systems from a configurable model. Such a system is viewed as a monolithic whole and the entire system is configured. However, in method engineering, we are dealing with relatively diverse phenomena: atomic/ compound methods, constructional/transformational methods, different methods for different stages of the life cycle [7]. It is not necessary that even after configuration is done, a full acceptable situated method is produced. Therefore, in contrast to system configuration, assembly plays an important role in situational method engineering. The basic techniques for building configurable methods and the basic configuration process can however be adopted from system engineering. According to [25] there are five main steps (a) establish scope (b) identify commonalities and variability (c) bind the variability by placing appropriate limits (d) exploit commonalities and (e) accommodate variability. In this paper we have shown that these steps can be seen in configurability for methods.

Regarding determining what is common [19] introduces a threshold. If the percentage of family members showing the feature is above this threshold then the feature is considered to be common. However, [25] is silent on how commonalities are determined. At this stage in our work, we have left this decision to the experience of the configurable method engineer. However, we intend to explore a mechanism for determining commonality in the future.

8. Conclusion

In this paper we extend the notion of configurability to method engineering. The approach adopted in this paper is shown in Fig. 4

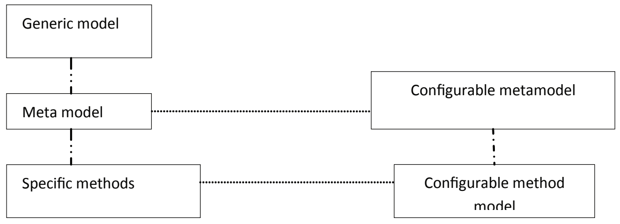

FIG. 4: INSTANTIATION OF CONFIGURABILITY USING GENERIC META MODEL

On the left hand side of the figure is the view of traditional method engineering [26]: a meta-model is an instantiation of a generic model and methods are instantiations of the meta-model. In order to facilitate configurability, we introduce configurability in the Meta model and consequently in the configurable method. The latter is an instantiation of the former.

We propose a two-part SME process of configuration and assembly. The former ensures that the methods being assembled are 'good' methods in the sense that individually they meet a part of the requirements of the situated method. The collection of methods to be assembled therefore provides a degree of assurance of yielding the situated method.

The main advantage of our approach is significant reduction in efforts to be put in by Method Engineers. Firstly, with the introduction of common and variable concepts, the method engineer has to focus only on the variable concepts as per situational requirements. Secondly, assembly process is mitigated until new concepts other than common and variable are required. This remarkably reduces the method engineer's efforts.

In future, we expect to lay down a criterion for defining commonality. This will be based on hierarchy of conceptual structures which can be instantiated by the CAME tool MERU [11] and then method knowledge will decide common and variable. An implementation of the configurability proposals made here is under way and we expect to verify it for configurability against a number of methods. Subsequently, we shall implement the full SME process as proposed here. Lastly we are extending MERU to support method configuration and assembly.

References

[1] C. Rolland, N. Prakash, "A proposal for Context-specific Method Engineering, Method Engineering Principles of Method Construction and Tool Support," *Brinkkemper S., Lyytenin K., and Welke R.J., (eds.), Chapman & Hall*, pp. 191-208, 1996.

[2] K. van Slooten, B. Hodes, "Characterizing IS Development Projects, Method Engineering Principles of Method Construction and Tool Support," *Brinkkemper S., Lyytenin K., and Welke R.J., (eds.), Chapman & Hall*, pp. 29-44, 1996.

[3] R. Deneckere, E. Kornyshova, B. Claudepierre, "Contextualization of Method Components," *Proc. RCIS*, pp. 235-246, 2010.

[4] F. Harmsen, S. Brinkkemper, J.L. Han Oei, "Situational method engineering for informational system project approaches," *Methods and Associated Tools for the Information Systems Life Cycle,* pp. 169-194, 1994.

[5] G. Grosz, C. Rolland, S. Schwer, C. Souveyet, V. Plihon, S. Si-Said, C. Ben Achour, C. Gnaho, "Modelling and Engineering the Requirements Engineering Process: An Overview of the NATURE Approach," *Requirements Engineering Journal*, Vol. 2, No. 3, pp. 115-131, 1997.

[6] N. Prakash, "On Method Statics and Dynamics, Information Systems," *Pergammon press,* Vol. 24, No. 8, pp. 613-637, 1999.

[7] N. Prakash, Goyal S.B. "Towards a Life Cycle for Method Engineering," *In Proceedings Eleventh International Workshop on Exploring Modeling Methods in Systems Analysis and Design (EMMSAD'07)*, pp. 27-36, 2007.

[8] Prakash, N. and S.B. Goyal, "Method architecture for situational method engineering," *In RCIS*, pp. 325-336, 2008.

[9] S. Brinkkemper, "Method Engineering-Engineering of Information Systems Development Methods and Tools," *in Information and software technology,* Vol. 38, pp. 275-280, 1996.

[10] S. Moaven et.al. "Towards an Architectural-Centric Approach for Method Engineering," *In IASTED conference on Software Engineering, Austria* pp. 74-79, 2008.

[11] D. Gupta and N. Prakash, "Engineering Methods from their Requirements Specification," *in Requirements Engineering Journal* , Vol. 3, pp. 133-160, 2001.

[12] J. Cameron, "Configurable development processes," *Communications of the ACM*, Vol. 45, No. 3, pp. 72-77, 2002.

[13] B. Fitzgerald, G. Hartnett and K. Conboy, "Customizing agile methods to software practices at Intel Shannon," *in European Journal of Information Systems*, Vol. 15, No. 2, pp. 197-210, 2006.

[14] M.L. Rosa, "Managing Variability in Process-Aware Information Systems", *Doctor of Philosophy, Faculty of Science and Technology Queensland University of Technology, Brisbane, Australia, 2009.*

[15] J. Coplien, D. Hoffman, D. Weiss, "Commonality and Variability in Software Engineering," *IEEE Software*, pp. 37-45, 1998.

[16] D.M. Weiss, C.T.R. Lai, "Software Product-line engineering: A Family based Sofwtare development Process," *Addison Wesley, 1999.*

[17] T.H. Davenport, "Putting the enterprise into the enterprise system," *Harvard Business Review* Vol. 76, No. 4, 1998.

[18] P. Soffer, B. Golany, D. Dori, "ERP modeling: a comprehensive approach," *Information Systems* Vol. 28, No. 6, 2003.

[19] M. Moon, K. Yeom, "An Approach to developing Domain Requirements as a Core Asset based on Commonality and Variability Analysis in a Product Line," *IEEE TSE,* Vol. 31, No. 7, pp. 551- 569, 2005.

[20] F. Karlsson, P.J. Ågerfalk, "Method Configuration: Adapting to Situational Characteristics While Creating Reusable Assets," *In Information and Software Technology* Vol. 46, pp. 619-633, 2004.

[21] K. Wistrand, F. Karlsson, "Method Components – Rationale Revealed," *In Advanced Information Systems Engineering 16th International Conference, CAiSE 2004, Riga, Latvia, June 7-11, 2004, Proceedings, A. Persson, J. Stirna, Eds. Springer-Verlag, LNCS 3084, Berlin,* pp. 189-201, 2004.

[22] J. Ralyté, C. Rolland, "An Assembly Process Model for Method Engineering," *In Advanced Information Systems Engineering, K.R. Dittrich, A. Geppert, M.C. Norrie, Eds. Springer, LNCS2068, Berlin,* pp. 267-283, 2001.

[23] T.W. Olle, J. Hagelstein, I.G Nacdonald, C. Rolland, H.G. Sol, F.J.M van Assche, A.A. Verrijn-Stuart, "Information Systems Methodologies A Framework for Understanding," *Addison Wesley, 1991.*

[24] N. Prakash, "Towards a Formal Definition of Methods," *Requirements Engineering Journal, Springer, Vol. 2, No. 1, pp. 23-50, 1997.*

[25] J. Coplien, D. Hoffman, D. Weiss, "Commonality and Variability in Software Engineering," *IEEE Software,* pp. 37-45, 1998.

[26] N. Prakash, "On generic method models," *Requirement engineering Journal11 (4), Springer,* **pp. 221-237, 2006.**

ABOUT THE AUTHORS

Prof. Daya Gupta is Professor in the Department of Computer Engineering, Delhi Technological University New Delhi, India and is currently the Head of the Department there. She has done M.Sc. Post M.Sc. (Computers Sc.) from IIT, Delhi, and PhD from Delhi University. She is a senior member of IEEE and a life member of CSI. Her research interests are in the field of requirement engineering, method engineering, information security, image characterization and software estimation. She has published several research papers in international journals and conferences and has chaired sessions and delivered invited talks at many national and international conferences.

Ms. Rinky Dwivedi has received the B.Tech. degree in Computer Science from Guru Gobind Singh Indraprastha University, Delhi, India and M.E. degree from Delhi College of Engineering, Delhi, India. She is now pursuing PhD from Delhi Technological University, Delhi India. Her areas of interests include Method Engineering, Software Methodologies and Agile Techniques .Rinky can be contacted by e-mail at rinkydwivedi@gmail.com.