Developing Families of Method-Oriented Architecture

Mohsen Asadi[1,2], Bardia Mohabbati[1,2], Dragan Gašević[2], Ebrahim Bagheri[2,3]

[1]Simon Fraser University, Canada,
[2]Athabasca University, Canada, [3]National Research Council Canada
{masadi, mohabbati}@sfu.ca, dgasevic@acm,org, ebagheri@athabasca.ca

Abstract. The method engineering paradigm is motivated by the need for software development methods suitable for specific situations and requirements of organizations in general and projects in particular. Assembly-based method engineering, as one of the prominent approaches in method engineering, creates project-specific methods by (re-)using method components, specified with method processes and products, and stored in method repositories. This paper tries to address the two challenges of assembly-based method engineering related to more effective: i) publication and sharing of method components; and ii) management of variability in software methods, which have many commonalties. In order to address these two challenges, we propose the concept of *Families of Method-Oriented Architectures*. This concept is built on top of the principles of service-oriented architectures and software product lines.

Keywords: Method engineering, Software Product Lines, SOA

1 Introduction

The increase in the complexity of software-intensive systems has urged for the integration of seminal approaches such as Object-modeling Technique (OMT) and Objectory to form integrated (plan-driven) and unified frameworks such as the Rational Unified Process (RUP). Integrated approaches typically target development of a vast variety of software applications, which increase the size of methods and make them become "cook-book" approaches. Recent critical literature reviews and comprehensive case studies have shown that such cook-book approaches do not work successfully for all circumstances [1]. Practitioners could potentially waste up to 35% of their effort by following the steps of standard development methods [3]. Moreover, the results of such studies reveal that the formal definition prescribed by a method in forms of stages and steps widely differ from the method actually being used [4]. These issues have motivated the software engineering research community to establish the *Method Engineering (ME)* [3] discipline. The ME community concentrates on the idea of providing an "adaptation framework whereby methods are developed to match specific organization situation" [1]. The most prominent ME approach is the *assembly-based method engineering* that creates a new method by assembling existing method components [6][16]. Despite the fact that ME has recently produced promising research results, there are still many open research challenges [1]. In this paper, we focus on two key challenges, namely:

1. The lack of a standard model for describing method components limits the opportunities of method engineers, teams and organizations to share, discover, and retrieve distributed method components. When a method engineer wants to create a new method from scratch or by adapting (extending/constraining) an existing method, a common approach is to try to reuse existing method components from the method repository. Therefore, method components need to be discovered and composed with other method components. Due to the lack of standards, method engineers are forced to reuse method components from the local proprietary repositories, without effective capabilities for retrieving method components in repositories of their collaborators. Moreover, with this limitation of method component sharing, business opportunities of organizations are also limited. In fact, they cannot easily publicize and offer the methods that they are specialized in, as (for profit) services.
2. In essence, organizations initially adopt a method for software development. Afterwards, components of the method may be subsequently added and gradually extended. Such extensions may be derived due to either the evolution of software development or various variations created for some specific method components. Some sources of these diversities may be differences among domains of systems under development (i.e., desktop application, web application, and real-time) or newly emerged software development approaches such as Model Driven Development, Component based Software Development as well as method types such as agile or plan-driven. Thus, there is the need for a systematic approach to manage variability of software methods and adapt software methods (families) that best suit the needs of a specific development context.

The first challenge has been already recognized in the literature [13][1] and some researchers have proposed to use of SOA and Web service standards and principles for dealing with the challenge [13][1]. To this end, the concept of *Method Services* was coined in analogy of the concept of services in SOA. In order to address both of the above challenges at the same time, we propose combining principles (Sect. 4 and 5) of Software Product Line Engineering (SPLE, Sect. 3.1) and Service-oriented Architectures (SOAs, Sect. 3.2) [12]. We use the method service notion for defining method components. We also propose leveraging SPLE with the goal of addressing the second challenge. Our key idea is to introduce a concept of method families, which share a set of common method components, and yet have effective tools for variability management (e.g., feature models). With the use of SPLE principles, we can allow for a systematic modeling of method families and for an automated process of specialization of method families where each family specialization satisfies requirements of a specific situation.

2 Motivating Example

In order to illustrate the challenges that are tackled in this paper, let consider an organization, which develops software systems in two distinct domains, namely, *information systems* (both desktop and web-based systems) and *real-time systems*. We consider that the organization has adopted a base method (e.g., RUP) for the entire

systems development process. The base method supposedly is a modular method and its method components are stored in a method repository. Moreover, the organization has employed different development approaches, including *code centric development, component based development*, and *model driven development*. Based on the scale and complexity of a project, the organization may follow different development policies such as *agile* or *plan-driven*. In addition, *contingency factors* such as time pressure, user involvement and project familiarity cause the source of diversity in method components. Furthermore, *human factors* (e.g., roles in the organization and their experience level) could be a source of variation points in the method activities. The organization might also intend to add more requirements for future variations of methods and integrate more project management method components in order to have a better support for project management and risk assessment tasks. As a response to the described circumstance, the organization requires to extend the base method using different method components. As a consequence, the complexity and variations of the base method are gradually increased in practice. This complexity leads to a limited sharing and management of lessons learned. Thus, there is a need to more effectively: 1) manage different variations of the base method that were observed and encountered in the previous projects and systematize the lessons learned; 2) anticipate further needs by considering all possible variations of the base method; and create a systematic method for adaptation of the base method considering the needs and requirements of the new development situations. Moreover, the organization, besides its own development projects, might also want to offer some consultancy services or partner with some other organizations based on expertise in method engineering. In such cases, the organization needs to have a standard method for publishing their competencies, so that other organizations can effectively discover and reuse such experience in similar development situations.

3 Background

3.1 Software Product Line Engineering

The SPLE paradigm aims at managing variability and commonality of core software assets of a given domain in order to facilitate the development of software-intensive products and to achieve high reusability [2]. SPLE empowers the derivation of different product family applications (aka, family members) by reusing the realized product family assets such as common models, architecture, and components. In this context, software assets are characterized by a set of *features* shared by each individual product of a family. The set of all valid feature combinations defines a set of product line members of the family. A valid composition of features is called a *configuration* which in turn is a valid software product specialization. The development of a software family is performed by conducting the *domain engineering lifecycle* in which the common assets, family reference architecture and the variability models are developed. Afterwards, in the *application engineering lifecycle*, the common assets are reused and variability models are configured to produce a family. *Feature modeling,* as a popular technique for modeling variability, is employed to

represent the variability and describe the allowed configurations of a software family. This technique is typically used in domain engineering to model an entire family based on the functional characteristics (aka features) that the family provides. Feature models formally and graphically define relations, constraints, and dependencies of software artifacts in a software product family. In essence, there are four types of relationships related to variability concepts in the feature model. They can be classified as: *Mandatory (Required), Optional, Alternative feature group and Or feature group*. Common features among various members of the family are modeled as mandatory features. In other words, mandatory features must be included in the description of their parent features and must be present in any final configuration. Optional features may or may not be included in a final configuration. Alternative features indicate that only one of the features from the feature groups can be opted.
Once a feature model for an entire family is in place, a process of configuration follows. Configuration is a process of selecting features needed for specific applications. Recently, the research community has proposed effective methods for staged configuration where each stage addresses a specific set of requirements in the application development process [11].

3.2 Method Oriented Architecture

Service-oriented computing (SoC) is a computing paradigm that promises flexibility and agility in the development of collaborative software systems. Service-oriented Architecture (SOA) is the main architectural style for realizing the SoC vision. SOA provides an underlying structure enabling for interoperability and communications between services. Web service, reusable and loosely-coupled components, are the best known materialization of SOAs [12]. Web services are built on well-defined standards such as Web Services Description Language (WSDL). Furthermore, the widespread adoption of Web service technologies provides open standards which increase accessibility and interoperability of distributed software services in a networked environment.

On the other hand, ME approaches are hindered by the lack of standards for describing the interfaces of components of methods. Moreover, reusable method components are restrained to be adopted locally by their providers in proprietary repositories. Indeed, the discovery and retrieval of reusable method components can significantly enhance rapid method construction and reuse. The ME community has already proposed the notion of Method Oriented Architecture (MOA) [1][13], which builds on and adopts SOA principles. Rolland proposed the MOA approach where Method as a Service (MaaS) is considered as an analogy to Software as a Services (SaaS)[1]. MOA aims at developing an ME approach, which elevates the accessibility of *method services* and facilitates their automated composition. In MOA, method services are described by method providers through WSDL documents. On the other hand, clients search and retrieve the required method services and compose them in order to create their own more complicated method service.

4 Method Services and Feature Modeling

As mentioned in the introduction section, to address the challenge of variability in method engineering, we intend to apply SPL principles and techniques especially feature modeling. In SPL, functionalities of a set of similar software systems and their visibilities are presented in a feature model in terms of features and variability points, respectively. Likewise, a set of similar methods (we call a family of methods) may have commonalities and variabilities with respect to functionality (i.e. activities). Therefore, a family of methods provides the means for capturing the commonalities (core assets) of all possible methods of a given domain and also addresses variability by covering a comprehensive set of dissimilarities between the methods. In our proposal for family development, distinguishable characteristics of a method mostly including functionalities of the method (i.e. activities) are represented using features. For instance, in our motivating example, one feature of the family is Use-Case modeling. The methods commonality and variability, in terms of their features, are represented in feature model. The development of a family of methods is performed by conducting the *domain engineering* lifecycle (developing feature model and implementing features), which is followed by the *application engineering* lifecycle (developing target method with configuring feature model). We should note that the feature model is only the representation for family characteristics and the variability relations and we need to link them to corresponding method implementations (i.e. method fragment). We use method services as well as MOA techniques (i.e. Method service discovery and composition) to implement features of a family. Therefore, we refer to our approach as development of families of method oriented architectures.

In order to clarify the difference between feature and method service, let us consider the process of method construction as a process of problem solving, in which the requirements model and the final method are considered as the problem space and the solution space, respectively. Since we intended to develop a range (i.e., family) of solutions (i.e. methods) which have common and variable parts, both the problem and solution spaces become more complex. By following SPLE principles, the family problem space (i.e., family requirements model) is decomposed and grouped into features which form a feature model. In other words, a feature intuitively represents sub-problems of the family problem space, and a feature model represents a hierarchical representation of the family space with variability. For instance, the problem space (feature model) of a described family method at the highest level is decomposed into (see Section 6) management, requirement engineering, development, and deployment sub-problems (features). On the other hand, method services form the solution space, in which one or more method services (sub-solutions), implement (solve) one or more features (sub-problems). From another point of view, features address what the properties of the solution are and method services represent the realization of those properties. Fig. 1 shows the *use case modeling* feature (one of the features of the feature model given in Section 6) and the corresponding *use case modeling* method service. As the figure shows, the method service represents how the modeling of use cases should be conducted. Also, a feature represents some functionality, which can be included in a method variant. One of the key concerns in method family engineering is the identification of method services for each feature and the binding of features onto method services. Then, in the application engineering

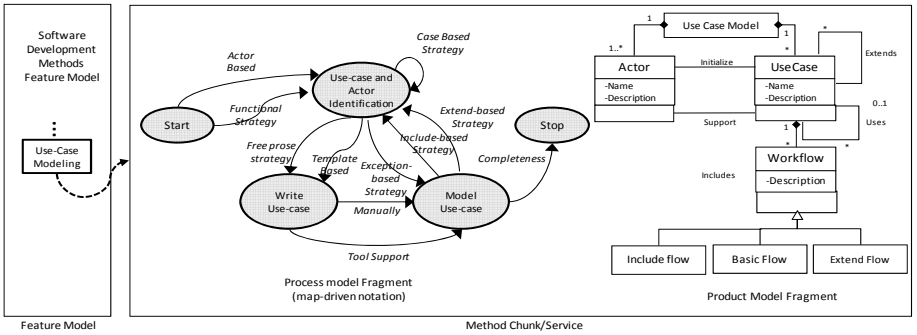

Fig. 1. The relation between use case modeling feature (on the left part) and its corresponding method services which define both process and product model for use-case (adapted from [16]).

lifecycle, method engineers select features from feature models corresponding to the requirements of the target method (i.e. feature configuration). Next, the method services bound to features in domain engineering are composed automatically and they form an initial method for application engineering. The initial method is adapted and improved until a suitable method is reached for the target problem and deployed.

5 Families of Method Oriented Architecture

Similar to developing software product lines, we propose two main lifecycles for method family engineering process, namely the *Method Domain Engineering* and *Method Application Engineering* lifecycles. Method Domain Engineering lifecycle is carried out one time for the whole family and develops the architecture of the method family, common assets, and variants. In this lifecycle, family features and their variability are modeled by a feature model and suitable method services corresponding to features (i.e. a feature implementation) are discovered and bound to the features. The method application engineering lifecycle develops a target method (i.e. a member of family) for a concrete application by configuring the feature model and assembling the method services related to the configuration. The method application engineering lifecycle is carried out every time a new method is required. The remainder of this section describes the main phases and activities of both lifecycles along with their associated product artifacts.

5.1 Method Domain Engineering

Method domain engineering aims at discovering, organizing, and implementing common assets of a method family. Moreover, determining the scope of a method family and describing the variability of the models is achieved during this lifecycle. The input of the lifecycle is domain knowledge relating to and describing the method family and the reusable assets, while variability models for the methods expressed

using feature models are the output of this lifecycle. Fig. 2 illustrates the phases and stages of the method domain engineering lifecycle.

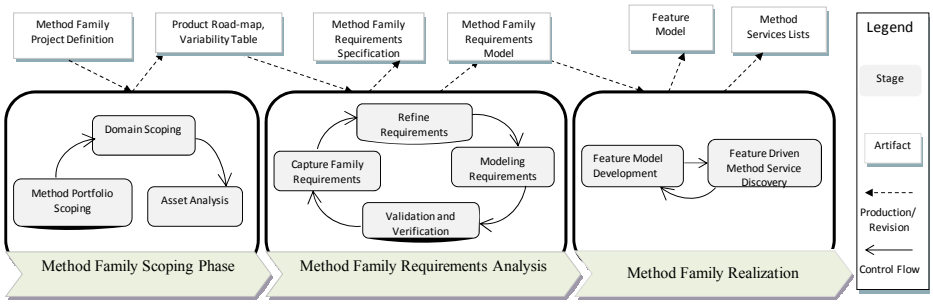

Fig. 2. The Method Domain Engineering Lifecycle

Method domain engineering starts with the *Method Family Scoping* phase which is a key phase for achieving economic benefits of a product line [2]. The Method Family Scoping phase aims at determining a set of products (Software Development Methods) which belong to the family. Scoping of the family is performed in the three stages [2]. The *Method portfolio scoping* stage is a high level domain analysis process and uses the market inputs on existing methods, and expert knowledge to derive a standardized description of a method product line, technical domains that are relevant to it, and the range of methods that shall be supported with the method family. It systematizes the method product information, identifies the main features of the product line and checks the consistency. With regard to features of a method family, the development approaches used in methods (e.g., Model Driven Development-MDD, and SOA), final application domains (e.g., Information System, Embedded Systems, and Ubiquities Systems), and method types (e.g., agile or plan-driven methodologies) are determined through this phase using the project documents.

Later, the *domain scoping* stage uses the basis provided in the previous stage and the expert inputs to identify and group the major functional areas in terms of technical domains which belong to the current method family. Moreover, the benefits and risks pertaining to the various domains are explored and documented. For instance, benefits and risks of employing MDD are identified. Finally, in the *asset analysis* stage, based on the preconditions established in the previous two stages, precise functionality of the method components that should be supported by the method family are described. This stage determines which assets should be developed for reuse (commonality) and which ones as project-specific (variability). The method engineer indicates the variable features (project-specific) belonging to the family, the type of the variables (e.g. logic, workflow), set of variants for the variable, and status of variants (open or close) [17]. The method product-line roadmap is produced as the output of this phase.

The *Method Family Requirements Analysis* phase aims at capturing requirements and developing a requirements model for the methods family. The family requirements model contains unique and unambiguous definitions for each requirement as well as the variability of the requirements. The phase receives the documents, stakeholders' viewpoints, and the product-line roadmap, and variability

ranges. Similar to typical software engineering procedures, we define *functional requirements* and *non-functional requirements* for methods. The functional requirements show the properties that the method should provide, such as work products and required activities; and non-functional requirements include properties that the entire or a large part of methods in the family should have such as smoothness of transition between activities, robustness, and scalability. The method family requirements are *elicited* and *documented*. The method engineer gets an agreement of developers (i.e., stakeholders in this case) on the method family requirements. Next, family requirements are *refined* through *decomposition, aggregation*, and *grouping*. Afterwards, in the modeling family requirements stage, techniques such as the map-driven technique [9] are applied to develop the family requirements model. The family requirements model includes the functional requirements and is represented as family requirements map. The progression activity analyzes the family requirements model and defines the requirements filling the gaps in the family requirements model. Finally, the method engineer verifies the completeness and coherence of the family requirements models as well as the level of satisfaction of the stakeholders' needs by using *Requirements verification* and *Validation* activities [9], respectively.

The goal of the next phase, *Method Family Realization* phase, is to identify common and variant features within the family and to model them with a feature model. Afterwards, the appropriate method services are discovered for each of the features. The feature model is developed by the *Feature Model Development* stage. That is, the common and variable functionalities of methods of the family are managed by representing them in a feature model. The method engineer starts from the requirements and analyzes the requirements, their granularity level and relationships, and then groups them into appropriate features. Moreover, the variability relations are identified between features. Additionally, nonfunctional requirements such as traceability and project management are analyzed and added to the feature model as features and their relations are also identified. Furthermore, the method engineer annotates the features with required information.

The requirements and feature family modeling phases produce the requirements model, requirements documents, and feature model of the method family. Feature family modeling, as described above, is followed by a *Feature Driven Method* for service discovery and selection. The stage of the feature-driven discovery is performed by considering each individual feature and their respective annotations. In essence, a feature annotation provides functional and non-functional keywords used to generate feature queries. The feature queries simply describe what the desired method services should be and how they should behave. In our current implementation, we adopted text-based approach to the discovery of method services. In evaluations of our current implementation of the feature-driven service discovery, we observed promising results in experimenting with the active service search engines while developing families of software services [15]. Since, MOA uses SOA standards for defining and publishing method services, we may expect similar results for discovering of method services. Given that there are no publically available repositories of method services, we are now developing a test collection of method services. In this process, we can easily leverage existing service repositories (e.g., Seekda already used in our implementation) for storing method services. Other

approaches can be leveraged in feature-driven service discovery such as logic-based approaches [14].

5.2 Method Application Engineering

Once the method domain model is created, then method engineers can take the method domain model and create different instances out of it based on target method requirements. We refer to this process as *method application engineering*.
Therefore, method application engineering aims to develop a method for a target situation (e.g. a member of the method family) by utilizing the reusable assets created in the domain engineering lifecycle. The input of the lifecycle is the project documents for the concrete method and the output is the method satisfying the requirements. It captures the final method requirements, selects the corresponding features from the feature model, and finally assembles the method services bound to the selected features. The application engineering process is illustrated in Fig. 3.

The *Application Method Requirement Analysis* phase aims to define the requirements of the target method. The documents related to the required method are the inputs and its requirements model and the requirement documents are the outputs. The documents related to the target method should include *definition* (specify the type of the project at hand), *domain* (specify the application domain of the target method), and *deliverable* (specify the artifacts that should be produced) [10]. The family requirements model and documents are utilized through this phase to produce the method application requirements.

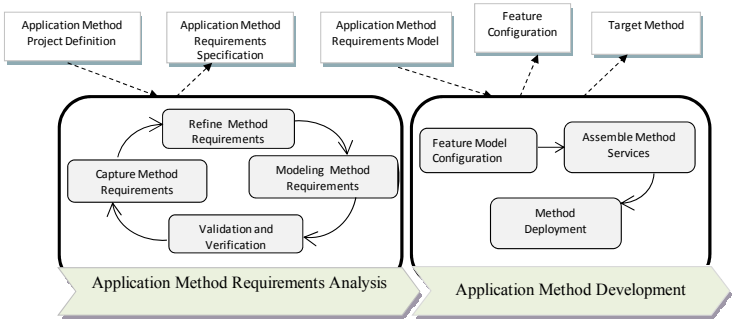

Fig. 3. Method Application Engineering Lifecycle

First, the method application requirements phase captures stakeholders' requirements and documents them. Then, method requirements are refined and clarified further and the agreement of stakeholders is achieved. Next, the method engineer develops the requirements model in the form of a requirements map. Moreover, non-functional requirements are utilized in the feature selection process. Finally, the method requirements are validated and verified to check the completeness and correctness of the method requirements. In all the activities of this phase, the family requirements model is used as a reference to facilitate the process of requirement analysis of the members of the method family. There is a possibility of

capturing requirements which were not captured in method family requirement analysis. The activities of this phase concentrate on one method application, so they do not deal with variability in the family.

The *Application Method Development* phase creates the target method by configuring the method family and delivers the final method configuration to the developers. The method *feature model configuration* stage aims to develop the method by selecting the most appropriate set of features from the feature model through a stage configuration process. It receives the method requirements and produces the corresponding feature configuration. The *stage configuration* process [11] starts from the feature model and carries out successive specializations to create the final configuration. That is, the staged configuration process would limit the space of the method family to the space most relevant for the current method that is being built. Through the staged configuration, the method engineer produces the final configuration. Since in method domain engineering, the method engineer might want to bind a list of method services that have the same interfaces (i.e. situation and intention) but different nonfunctional properties defined in descriptors of method services, the final method service for each feature is selected from the list of alternative method services. The output of the stage is the set of the features (mandatory and optional) as well as their corresponding method services.

If the selected method services (features) do not cover all the requirements of the target method, the new method services for the remaining requirements are discovered in some other repositories or developed from scratch. After the method engineer makes sure that all required method services (features) have been gathered, he/she starts the composition of method services (features) via the *Assemble Method Services* stage. The selected features are divided into functional (e.g., requirement elicitation, use case modeling, and developing design model) and non-functional features (e.g. quality assurance, project monitoring, and traceability checking). First, the method services are orchestrated and the necessary adaptation and mediation are conducted. Then, a decision about the location of method services within a large scope (like quality assurance) is made. After creating the target method, the verification/validation task is done by the method engineer to check whether the method is free from defects and if the target method meets all requirements established in the requirements phase. Moreover, the completeness of the method is verified by a *completeness* task. Finally, the method is deployed to the stakeholder environment by preparing method documents, training developers, and supporting staff through the execution of the method.

6 Case Study

In this section, we represent our motivational example from Section 2 by following our proposed approach described in the previous section. Due to the space limitation, we only explain the domain method engineering lifecycle, which comprises *Product Line Scoping*, *Family Requirement Analysis, Feature Modeling and Feature-based Method Service Discovery and Selection*.

Product Line Scoping: By completing the activities of the product line scoping phase, we identified the criteria which specify the product line boundaries, the main functionality area, and core assets of the method family. Table 1 shows a part of the product line scoping results. One of the major functionality areas distinguished in the domain scoping by all variations of the method is the support for a generic development lifecycle. For instance, unit testing is a core asset in the method family.

Family Requirement Analysis: Functional and non-functional requirements with their commonalities and variability are captured and documented separately. Table 1 shows a part of requirements categorized based on their types. Functional requirements include activities and work products that should be supported with family method. The base method of the organization is explored to discover more detail requirements. The family requirements model is created first by using map-driven approach [6] and then verified and validated. Due to the space limitation for this paper, the requirements model is omitted from the paper.

Feature model Development – based on the family requirements model defined in the previous phase and the existing basis method in the organization, features and their corresponding relations are identified and modeled. The part of feature model designed for target organization is depicted in Fig. 4. Features show the method services required for the family and they can be considered as interfaces for representing method services in the family.

Feature-driven Method Service Discovery and Selection: the next step after feature model development is the discovery of method services. The aim is to find and select among available methods services, which can satisfy desired functional and non-functional requirements of the method for specific situations. As we described earlier, we consider each feature and their associated annotations as queries for method components stored in method repositories. In method service discovery, we assume that the method components, described by WSDL, are available and accessible through either the proprietary method repositories of the organization or public repositories provided by third-parties. Thereby, organizations can publish and share their method chunks as services. Although there are on-line repositories such as Open Process Framework (OPF) [20], available reusable method components are not accessible through standard interfaces. Moreover, there are no facilities to search and discover such available methods. Accordingly, in the process of discovery and selection, the proprietary method repository of the organization is initially used to method services. In case that some of the features are not associated with the organization's services, search queries are broadcasted to the public method repositories.

The Feature Model Plugin (http://gsd.uwaterloo.ca/projects/fmp-plugin/), available for Eclipse environment, is utilized and extended as tool support for modeling and configuring method family. It supports cardinality-based feature modelling, specialization of feature diagrams and configuration based on feature diagrams. Our method chunk service repository is based on the publicly-available Seekda (http://seekda.com) service repository. Our current implementation of feature-driven service discovery is described in [15].

Table 1. Product line scoping and family requirements analysis outcomes after applying the proposed method on the motivational example. It is important to notice that the table does not give all items identified these phased, but some of the most notable examples.

	Phase	Identified work Product and domains
Product Line Scoping	Portfolio	• *Application properties* – application domain (Information systems, Real-time), application type (intra-organization, Organization-customer, inter-organization), source system (it can either use legacy system or does not have system code). • *Development Approach* – systems can be developed by following multiple approaches such as Component based Development, Model Driven Development, or Test Driven Development. • *Human Factors* such skill level includes beginner, medium, and expert (i.e., analyst, designer, developer, and, tester). • *Contingency Factors* –user involvement, project familiarity, project scale and complexity, innovation level, and project dependency. • *Project Management* – monitoring, risk management, configuration and change management, postmortem reviewing, metric management, human resource management.
	Domain	• Generic software development lifecycle (requirement engineering, analysis, design, development, deployment), reusability, management (risk, people), maintenance, test model, implementation models, design model, and Application Technology (Include Data-base, and GUI, is distributed).
	Asset	• Functional requirement engineering, non-functional requirement engineering, behavioral analysis, structural structure, functional analysis, feasibility study, architecture design, project planning, test case development, unit testing, and risk management.
Family Requirement Analysis	Functional Requirements	• *Common* – Specification in high level abstraction, covering generic software development lifecycle, manage and monitor the project, capture requirements, model requirements, validate requirements, defining the infrastructure of system, and plan the project. • Variables- Goal-based requirement extraction, consider review sessions (product and plan review), having stand up meeting, having lightweight design process, formal verification on each abstraction level, concurrency, configuration of software and hardware, having platform independent models, having platform specific models, component identification, component specification, component interaction, component assembly, and PIM and PSM synchronization.
	Functional Requirements	• Common – iterative process, incrementally development, traceability to requirements, clear separation of concerns, smooth transition between activities, and method flexibility. • Variables - semi automatic refinement between abstraction level, method scalability, lightweight process, and formal checking.

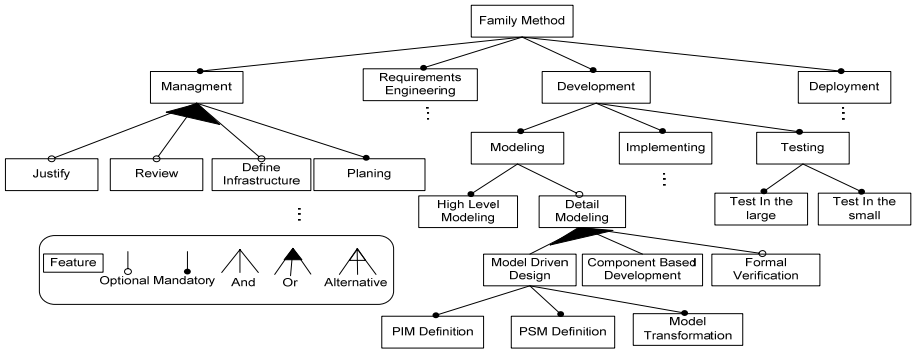

Fig. 4. A sample feature model of a family of software development methods

7 Related Work

ME defines techniques and approaches for constructing and/or adapting the methods. The most prominent sub-area of ME, *Situational Method Engineering* (SME), proposed by Welke et al [5] is concerned with the creation of methods 'on-the-fly' (i.e. construct or adapt a method according to situation of the project at hand). The ME approaches are classified by Ralyte et al [6] as: *Ad-Hoc* (i.e. Method created from scratch); *Extension-Based* (i.e. Method is created by extending an existing method [6]); *Paradigm-based* (an existing meta-model is adapted, instantiated, or abstracted to create a new method [6]); and *Assembly-based* (a method is created by reusing existing method components [7][16][25]). These approaches mostly focus on reusability and modularity principles. Besides this classification, Karlsson et al. [8] proposed the *Method Configuration* approach (more general than extension based) in which a target method is created by adding/removing elements and features. They concentrate on variability management and reusability. All mentioned approaches are based on one or more of the following principles - meta-modeling, reuse, modularity, and flexibility. Our proposed approach is similar to the assembly-based and method configuration by following of the modularity, reusability, and variability principles. However, our approach enables for a higher degree of reusability by leveraging SOA principles and for a more systematic variability management by employing SPLE principles (As shown in software engineering SPLE increases reusability [24]).

Gonzalez-Perez [20] explained the benefits of ISO/IEC 24744 meta-model for both method specification and enactment and proposed a product-centric approach to developing a new methodology. Aharoni et al [22] enriched the ISO/IEC 24744 for creating and tailoring methods through an approach called Application-based Domain Modeling (ADOM). The approach is based on the layered framework including application, domain, and language. The domain (methodology) layer contains different method concepts as well as the specification of their exact usage situation. The application layer, called endeavor layer, includes specific method components and situational methods, which are created based on domain model terminology, rules, and constraints. The language layer defines any modeling language that can be used for describing meta-models and method components. Our approach differs from these approaches in using variability modeling language (i.e. feature modeling) and software product line principles. Moreover, we provide a reference architecture for a whole family which eases configuring and developing methods. Additionally, we use a new concept for method component (i.e., method service), which utilizes standards in SOA to improve discovering and reusing method components.

Recently MOA [13][1] was proposed which empowered the assembly-based method engineering principles with a standard for describing method components (in terms of method service) and with service discovery principles for finding distributed method components. Our approach also utilizes MOA to describe and discover method services corresponding to the features of a method family.

8 Discussion

Two main issues regarding the proposed approach are validity and cost-benefit analysis of the approach. For both issues, it is required to conduct an empirical study. We did a case-study in which we explained the steps of the method. However, it cannot completely ensure the validity of our approach, its benefits and limitations. In order to clarify these issues in our method, we make our argument based on analogy between software and methods as proposed by Osterweil [23]. Therefore, our assumption is "software processes are software too" [23]. Considering this analogy, we can adopt similar approaches and techniques used in software engineering for solving existing problems in method engineering. As we see, the method engineering community proposed MOA inspired from SOA to deal with the lack of standard for defining the method fragment interfaces [1]. But, to use analogy as a viable strategy for solving a problem in method engineering, referred to as the target domain, we need to identify the corresponding construct in software engineering (source domain) and define a mapping schema. For example, in method engineering, the method fragment notion (called method service) is mapped to service notion in SOA and method notion is mapped to software service. Hence, we can use SOA principles and have benefits of SOA in the MOA domain. The other problem method engineers deal with is variability in the base method and configuring the method based on the target project for which some approaches have been proposed [6][8]. On the other hand, software variability is a well-known problem in the software engineering community and many techniques and approaches have been proposed like feature modeling to manage the problem and various success stories in using product families and associated techniques have been reported. As an example, Clements and Northrop reported that Nokia was able to increase its production capacity for new cellular phone models from 5-10 to around 30 models per year, which alleviated Nokia's main challenge being the high pace of market demand and customer taste change [24]. These results ensure both validity and benefits of software families. Therefore, we tried to make an analogy between software family and method base and coined the notion of family of methods. We mapped the features to the method fragment interfaces and handled the variability in base method and configuration problem according to the target project requirements. As a result, we expect similar benefits to be reaped within the method engineering domain. We are also aware of the cost of creating family or reengineering current methods into method family (i.e., creating a method feature model), but for long term the benefits that will be achieved can recompensate these costs as happened in the broader software engineering practice.

9 Conclusion and Future Work

In this paper, we have presented an approach for developing families of software developments methods. We exploited the notion of method services to facilitate the discovery of distributed method components. Such discovered method components can be used as an implementation for both sets of common and variable method assets of a family of methods. The proposed approach makes use of feature modeling to

manage variability of method families. Managing and modeling variability enables for a more effective method construction and for a more systematic method reuse. We believe that the described concept of families of method-oriented architectures may not be entirely feasible now, due to the lack of a complete method sharing ecosystem, but with the growing interest for services economy, more attention to such ecosystems can easily be envisioned to appear soon. Thus, our approach is a small step towards making this vision possible. The adoption of widely used SOA standards helps in publishing and sharing method components. Furthermore, organizations will be able to take the advantages of distributed architectures to design, implement, execute and reuse available method components. Last but not least, the long term goal is to enable different organizations and enterprises to publish, advertise, discover and reuse their methods components.

While the paper proposed a methodology for the combined use of SPLE and SOA principles in method engineering, the contribution of this paper deserves to be considered in a broader context of its implications. As already demonstrated in the previous research [18], transforming configured feature models into workflow and service composition languages is possible. Thus, the combined use of SOA and SPLE enables for leveraging existing workflow engines (e.g., BPEL) in management and execution of software projects. Moreover, with such an executable representation of methods as workflows, one can also expect an increased compliance of projects with the steps defined by methods. As workflow management provides also best practices (i.e., workflow patterns), the combined use of workflows with software methodologies might lead to further benefits such as improved parallelization of some stages. With representation of method components as services, tracking of the project progress could also be improved, while the invocation of method services can explicitly be associated with the other tools used in different method stages.

As future work, we intend to provide a more comprehensive evaluation of the proposed approach by developing a collection of method service to be used for experimentation. We aim to extend our approach from different perspectives to reduce the manual intervention needed in the final method development. We plan to use ontology-based representation for feature models to automate consistency checking of features in method families as described in [18]. Furthermore, we intend to extend the feature modeling language to allow method engineers to add concepts of domain ontologies for annotation of features. This will consequently be used for advanced ontology-based discovery and composition of method services [14]. Currently, we are developing an environment that supports our proposed process. The environment will include the modeling of method families, annotation of feature models, discovering of method services, stage configuration of feature models, and deployment to standard workflow management engines.

References

1. Rolland, C.: Method engineering: towards methods as services Software Process: Improvement and Practice, 14(3), 2009, pp. 143-164.
2. Schmid, K.: A comprehensive product line scoping approach and its validation. In *Proc. of the 24th international Conference on Software Engineering*, 2002, pp. 593-603.

3. Harmsen, A.F.: Situational Method Engineering. Moret Ernst & Young, Utrecht, 1997.
4. Lings B, Lundell B.: Method-in-action and method-in-tool: some implications for case. In Proc. 6th Int'l Conference on Enterprise Information Systems, 2004, pp.623-628.
5. Welke R.J., Kumar K.: Method Engineering: a proposal for situation-specific methodology construction. In W.W. Cotterman & J.A. Senn (Eds.) Wiley, 1992, pp. 257-268.
6. J. Ralyté, R. Deneckére, C. Rolland.: Towards a generic model for situational method engineering, in: Proceeding of CAiSE2003, 2003, pp. 95-110.
7. Ralyte, J., Rolland, C.: An assembly process model for method engineering. In Proc. of the 13th Int'l Conf. on advanced information systems engineering, 2001, pp. 267-283.
8. Karlsson, F., Gerfalk, P. J. A.: Method configuration: adapting to situational characteristics while creating reusable assets. Inf. and Soft. Technology. 46 (9), 2004, pp. 619-633.
9. Ralyte, J.: Requirements definition for the situational method engineering. In Proc. of the IFIP WG8.1 Working Conf. on Eng. Inf. Sys.in the internet context, 2002, pp. 127-152.
10. Coulin, C., Zowghi, D., Sahraoui, A.E.K.: A Lightweight Workshop-Centric Situational Approach for the Early Stages of Requirements Elicitation in Software Systems Developme. In Proc. of Workshop on Situational Requirements Eng. Processes, 2005.
11. Czarnecki, K., et al.: Staged Configuration through Specialization and Multi-level Configuration of Feature Models. Soft. Process: Improvement & Prac., 10(2), 2005, pp 143-169.
12. Tsai, W.: Service-oriented system engineering: a new paradigm. Service-Oriented System Engineering, In Proc. IEEE Int'l Workshop on Service-Oriented Sys. Eng. 2005, pp. 3-6.
13. Deneckère, R., Iacovelli, A., Kornyshova, E., Souveyet, C.: From Method Fragments to Method Services. In Proc. 13th Int'l Conf. on Exploring Modelling Methods for Systems Analysis and Design., 2008, p. 81-96.
14. Klusch, M.: Semantic Web Service Coordination. CASCOM: Intelligent Service Coordination in the Semantic Web, 2008, pp. 59-104.
15. Mohabbati, B., Kaviani, N., Lea, R., Gašević, D., Hatala, M., Blackstock, M.: ReCoIn: A Framework for Dynamic Integration of Remote Services in a Service-Oriented Component Model, In Proceedings of the 2009 IEEE Asia-Pacific Services Comp. Conf., 2009.
16. Mirbel, I., Ralyte J.: Situational method engineering: combining assembly-based and roadmap-driven approaches. Requirements Engineering 11(1): 58–78, 2006.
17. Kim, S., Min, H.G., Her, J.S., Chang, S.H.: DREAM: A practical product line engineering using model driven architecture," In Proc. Int'l Conf. on Information Technology and Applications, 2005, pp. 70-75.
18. Montero, I., Pena, J., Ruiz-Cortes, A.: From Feature Models to Business Processes. In Proc. of the IEEE Int'l Conf. on Services Computing Vol. 2, 2008, pp. 605-608.
19. Bošković et al. (2010)Automated Staged Configuration with Semantic Web Technologies, International Journal of Software Engineering and Knowledge Engineering (in press)
20. OPEN Process Framework (OPF) Web Site, http://www.opfro.org/
21. Gonzalez-Perez, C.: Supporting Situational Method Engineering with ISO/IEC 24744 and the Work Product Pool Approach. SME: Fundamentals and Experiences. 2007, pp. 7-18.
22. Aharoni, A., Reinhartz-Berger, I.: A Domain Engineering Approach for Situational Method Engineering. Proceedings of the 27th ER. Springer-Verlag, Spain 2008, pp. 455-468.
23. Osterweil, L.: Software processes are software too. Proceedings of the ICSE. pp. 2-13IEEE Computer Society Press, Monterey, California, United States (1987).
24. Clements, P., Northrop, L.M.: Software product lines, visited June 2010, http://www.sei.cmu.edu/programs/pls/sw-product-lines 05 03.pdf (2003).
25. Asadi, M., Ramsin, R.: Patterns of Situational Method Engineering, In *Software Engineering Research, Management and Applications (SERA) 2009*, R. Lee, N. Ishii (Eds.), SCI 253, Springer, 2009, pp. 277-291.