Developing semantically-enabled Families of Method-oriented Architectures

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

[1]Simon Fraser University, Canada,
[2]Athabasca University, Canada,
{masadi, mohabbati, mhatala}@sfu.ca, dgasevic@acm.org, ebagheri@athabascau.ca

Abstract

Method Engineering (ME) has emerged with the aim of improving software development methods by creating and proposing adaptation frameworks whereby methods are created to provide suitable matches with the requirements of the organization and address project concerns and fit with specific situations. Therefore methods are defined and modularized into components stored in method repositories. Retrieved methods can be integrated and adapted to form a coherent method by means of assembly processes. The assembly of appropriate methods depends on the particularities of each project, and rapid method construction is inevitable in the reuse and management of existing methods. The ME discipline aims at providing engineering capability for optimizing, reusing and ensuring flexibility and adaptability of methods; however, there are three key research challenges which can be observed in the literature: 1) the lack of standards and tooling support for defining, publishing, discovering and retrieving methods which are only locally used by their providers without been largely adapted by other organizations; 2) dynamic adaptation and assembly of methods with respect to imposed continuous changes or evolutions of the project lifecycle; and 3) variability management in software methods in order to enable rapid and effective construction, assembly and adaptation of existing methods with respect to particular situations. Hence, to address these challenges, we propose semantically-enabled families of method-oriented architecture by applying service-oriented product line engineering principles and employing Semantic Web technologies.

1 Introduction

The increase in the complexity of software-intensive systems has urged 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 the development of a vast variety of software applications, which increase the size of methods and make them become "cook-book" approaches. The recent critical literature reviews and comprehensive case studies have shown that such cook-book approaches do not work successfully in all circumstances [1]. Practitioners could potentially waste up to 35% of their efforts 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 the 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 *assembly-based method engineering* that creates a new method by assembling existing method components [6][26]. 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 the following two key challenges:

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 reusing existing method components from method repositories. 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 the repositories of their collaborators. In addition to 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 the 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 differ between domains of systems under development (i.e., desktop applications, web applications, and real-time systems) or newly emerging software development approaches such as Model Driven Development, Component based Software Development as well as method types (e.g., 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 already been identified and discussed in the literature, and some researchers have proposed the use of SOA and Web service standards and principles in order to deal with this challenge [18][1]. To this end, the concept of *Method Services* was coined in analogy to the concept of services in SOA. Although the notion of method service provides a standard for describing and publishing method components, method engineering lacks techniques for discovering proper method services for a situation at the hand. On the other hand, the second challenge has also been partially addressed by some techniques, which propose to capture variability and representing variability within methods [7][8]. However, situational characteristics, which are important for choosing a proper method component from a set of method components, are missing in these approaches.

In order to address both of the above challenges at the same time, we propose to combine principles from Method-oriented architectures (Sect. 2.1), semantically-enabled Service-oriented Architectures (Sect. 2.2), and Software Product Line Engineering (Sect. 2.3). To address the first challenge, we employ the method service notion for defining method components and incorporating additional semantics and meaning to method services (Sect. 3.3), which enables more informed-search throughout the available method services. We utilize the Semantic Annotation Web Service Description Language (SAWSDL [9]) with the aim of defining method services and annotating method services with relevant semantic information. Furthermore, we propose the leveraging of the 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 techniques and tools for variability analysis and management (e.g., feature modeling). To this end, we extended and employed feature modeling language (Sect. 3.2), the most well-known variability modeling language in SPLE, to represent variability in a family of methods. Hence, our proposed approach allows a systematic modeling of method families and helps to automate the process of specialization of method families where each family specialization satisfies the requirements of a specific situation (Sect. 4).

2 Background

2.1 Method-Oriented Architecture

Method Engineering (ME) processes call for an appropriate understanding of the dimensions of method development. The ME approaches are hindered by the lack of standards for describing interfaces of components of methods. Moreover, reusable method components are restrained by only local adoption by their providers in proprietary repositories. Indeed, the discovery and retrieval of reusable method components can significantly enhance rapid method construction. Hence, to address these research challenges, the ME community has already proposed the notion of MOA [1][18], which builds upon and adopts SOA principles. In [1], Rolland proposed the MOA approach where *Method as a Service* (MaaS) is considered in analogy to Software as a Service (SaaS). MOA aims to develop a ME approach, as the foundation for method-oriented computing, to provide a way to recognize a portfolio of existing methods into self-describing elements (services) which are publishable and accessible through standard well-defined interface and distributed over the network. This elevates the accessibility of *method* components and facilitates their automated composition, i.e., assembly and constructing situation-dependent methods to guide the development of applications. In recent works [1][18] toward MaaS, method services are described using the Web Services Description Language (WSDL). The WSDL document describes a Web service on the syntactic level. Nevertheless, it is indispensable to enrich the WSDL corpora by semantically rich annotations of method components (i.e., method chunks) descriptions in order to automate (partially or fully) tasks of method service discovery, composition, and invocation [17].

2.2 Semantically-Enabled Service-Oriented Architecture

Service-Oriented Computing (SOC) is an emerging 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 interoperability and communication between services. Web services, as reusable and loosely-coupled components, are the best known materialization of SOA [17]. Web services are built on well-defined interfaces. Furthermore, the widespread adoption of Web service technologies provides open standards, which increase accessibility and interoperability of distributed software services in network environments.

To support the automation of service discovery, composition and monitoring, many approaches have been proposed. In this paper, we refer to the notion of Semantic Web Services (SWSs). Recently, the W3C consortium proposed the Semantic Annotation of WSDL (SAWSDL) language [9], which is a standard enabling the semantic web services' description. Some other submissions for the SWS standards are Web Ontology Language for Services (OWL-S) [10], Web Services Description Language Semantic (WSDL-S) and Web Services Modeling Ontology (WSMO) [11]. The machine processable semantics of SWS are captured through the annotation of service description elements with concepts defined in externally maintained semantic models (i.e., ontologies defined as formal and explicit specifications of shared conceptualizations). SAWSDL enables WSDL descriptions to be enhanced with semantic descriptions of functional, behavioral, informational, and non-functional aspects of services. SAWSDL has numerous advantages, but one of the key ones is that SAWSDL is agnostic on the knowledge representation formalism that one adopts for semantic models, i.e., semantic models can be defined in languages such as Unified Modeling Language (UML), Ontology Web Language (OWL), or Web Service Modeling Language (WSML).

2.3 Software Product Line Engineering

The SPLE discipline aims at managing variabilities and commonalities of core software assets 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 starts with conducting the *domain engineering* process (i.e., family development), which is followed by the *application engineering* process (i.e., family specialization and configuration).

Feature modeling, as a distinctive and widely used technique for modeling variability, is employed to represent variability and describe the permissible and valid configurations of a software family. This technique is typically used in domain engineering in order to model the entire family. Feature models define relations, constraints, and dependencies of software artifacts in a software product family. There are four types of relationships related to variability concepts in a feature model: *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 presented 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 used. 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 proposed methods for staged configuration where each stage addresses a specific set of requirements [16].

3 Integrating Semantic Web Services and SPLE for Method Engineering

This section introduces the underlying ideas for integrating the principles of semantic SOAs and SPLE for developing families for method services[1]. First, we describe how to adapt semantic Web service technology to ME needs, particularly by focusing on the problem of how to describe semantic method components to become method services, which are implemented as Web services to address the first challenge. Next, we outline our proposal for applying and integrating the main technique used in SPLE (i.e., feature modeling) and method services in MOA for managing variability in ME. Next, the idea of using the Semantic Web for enabling the semantic discovery of method services for features is introduced. Finally, based on the proposed formalism for the semantic method service and the semantically enabled feature model, we describe and demonstrate how method services for features can be discovered and retrieved by utilizing a standard query language.

3.1 Semantically-enhanced MOA

This section presents how our proposal employs Semantic Web Services (SWSs) in order to address the first challenge and cater standard models to describe methods as a service. As mentioned before, SWSs are proposed as the means for automating common tasks involved in Web services [19]. Discovery, composition and invocation can be achieved with higher levels of automation when Web services are supplemented with semantic descriptions of their properties. The lack of machine-readable descriptors is only one of the key challenges, which has to be addressed in order to provide a certain level of automation for method services. Our proposal for the *semantically-enhanced MOA* is fully built on top of SAWSDL specification.

To illustrate the use of SAWSDL for the representation of method services, we have made use of the services given in [1]. In Figure 1, a method service called *ImproveRoleService* is described as a semantically-annotated Web service, which can improve the modeling of associations between two classes by partitioning classes based on a declared association relationship. The UML models, as input and output of the method service, are represented in the UML XML Metadata Interchange (XMI) format.

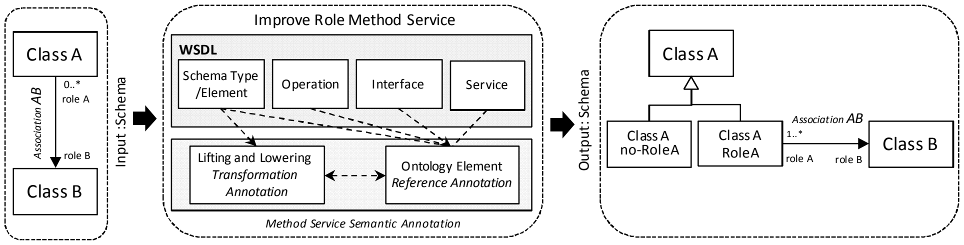


Figure 1: Improve Role Method service by semantic annotation [1].

According to the specification of SAWSDL, there are two types of annotation in SAWSDL, namely, *reference annotation* and *transformation annotation*. Basically, a reference annotation describes an element of WSDL with an ontology concept by using the *modelReference* attribute (e.g., Figure 2: lines 9, 11, 13, and 19). The *modelReference* attribute contains a value of the URI of the concept used for the annotation.

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <wsdl:definitions
3.   xmlns:sawsdl=http://www.w3.org/ns/sawsdl
       ⋮
4.   <wsdl:types>
5.   <xs:schema targetNamespace="ImproveRoleService">
6.     <xs:complexType name="ImproveRoleInput">
7.      <xs:sequence>
8.        <xs:element name="InputSchema„
9.            sawsdl:modelReference="http://example.org/ontology/UML/UMLOnto#XMIDocument"/>
10.       <xs:element name="ClassNameA“
11.           sawsdl:modelReference="http://example.org/ontology/UML/UMLOnto#XMIdref"/>
12.       <xs:element name="ClasswithRoleA“
13.           sawsdl:modelReference="http://example.org/ontology/UML/UMLOnto#XMIdref"/>
14.       <xs:element name="ClassWithNoRoleA" type="xs:string"/>
15.       <xs:element name="ClassWithRoleA" type="xs:string"/>
16.     </xs:sequence>
17.     </xs:complexType>
18.    <xs:element  name="ImproveRoleOutput"
19.         sawsdl:modelReference="http://example.org/ontology/UML/UMLOnto#XMIDocument"/>
20. </xs:schema>
21.  </wsdl:types>
22.  <wsdl:message name="ImproveRoleInput“>
23.    <wsdl:part name="parameter" type="xs:ImproveRoleInput"/>
24.  </wsdl:message>
25.  <wsdl:message name="ImproveRoleOutput“>
26.   <wsdl:part name="parameter" type="xs:ImproveRoleOnput"/>
27.  </wsdl:message>
28.  <wsdl:interface name="ImproveRole">
29.   <wsdl:operation name="ImproveRoleAction">
30.     <wsdl:input message="ImproveRoleInput"/>
31.     <wsdl:output message="ImproveRoleOutput"/>
32.   </wsdl:operation>
33.  </wsdl:interface>
34.  <wsdl:binding name="ImproveRoleBinding" type="ImproveRolePortType"> ... </wsdl:binding>
35.  <wsdl:service name="ImproveRoleService“ interface ="ImproveRole">
36.    sawsdl:modelReference="http://example.org/ontology/Method/MethodChunkOnto#MethodChunk">
37.  </wsdl:service>
       ⋮
38. </wsdl:definitions>
```

Element Declaration
Type Definition
Interface
Binding
Service
WSDL Description

Figure 2: WSDL description with semantic annotation of the improve role method service. Semantic model can be attached to the method service WSDL documents.

Figure 2 (line 36) also shows that the method service interface is annotated with a model reference which refers to the MethodChunk semantic concept in the Method Chunk ontology (MethodChunkOnto). Functional and non-functional semantics of a method service are expressed in this ontology. These semantics enrich the method service description to describe what the method service is able to offer along with the details of the method service implementation.

Figure 3 provides a fragment of *MethodChunkOnto*. For instance, the ontology specifies that the method chunk intention is to improve the modeling of associations using a hierarchical strategy. Moreover, it is applicable for modeling class diagram associations.

A transformation annotation defines how XML-based data are transformed into semantic models (so-called *lifting*). The reverse transformation is called *lowering*, which defines how the data are transformed from semantic models into XML-based data. The contents of the lifting or lowering transformation are omitted for the sake of simplicity.

Using this formalization, the method service along with its related concepts can be semantically defined in SAWSDL and OWL, respectively and then published. Later, when a specific method service needs to be found, the semantic discovery approaches are applied that provide more appropriate method services.

```
xmlns:ex="http://example.org/ontology/Method/MethodChunkOnto#MethodChunk"
<owl:Class rdf:ID="MethodChunk"/>
   ⋮
<owl:ObjectProperty rdf:ID="hasDescriptor">
  <rdfs:range rdf:resource="#Descriptor"/> <rdfs:domain rdf:resource="#MethodChunk"/>
</owl:ObjectProperty>.
   ⋮
<ex:MethodChunk rdf:ID="MethodChunk_ImproveRole">
  <ex:hasDescriptor>
    <ex:Descriptor rdf:ID="Descriptor_ImproveRole">
      <ex:hasReuseContext>
        <ex:ReuseContext rdf:ID="ReuseContext_ImproveRole">
          <ex:ReuseContextDescription xml:lang="en">applicable Class Diagram association modeling
          </ReuseContextDescription>
       </ReuseContext>
      </hasReuseContext>
      <ex:hasReuseIntention>
        <ex:ReuseIntention rdf:ID="ReuseIntention_ImproveRole">
          <ex:ReuseIntentionDescription xml:lang="en">Improve modeling of association using hierarchical
                                                      strategy
          </ReuseIntentionDescription>
        </ReuseIntention>
      </hasReuseIntention>
    </Descriptor>
  </hasDescriptor>
</MethodChunk>
```

Figure 3 A fragment of the MethodChunk ontology (MethodChunkOnto), expressed in OWL, is referenced as a model reference within the method service description given in Figure 2 (line 36).

3.2 Method Services and Feature Modeling

As mentioned earlier, to address the challenge of variability management in method engineering, we intend to apply SPLE principles and techniques especially feature modeling. In a software product line (i.e., a family of products), functionalities of a set of similar software systems and their variabilities can be described and presented by feature models in terms of features and variability points, respectively. Likewise, a set of similar methods (hereafter referred to as a family of methods) may have commonalities and variabilities with respect to the functionalities (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, features presents distinguishable characteristics of a method, mostly including functionalities of the method (i.e., activities). For instance, Use-Case modeling can be considered as a feature of a family. The methods commonality and variability, in terms of their features, are represented in a 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 the method by configuring the feature model). It should be noted that a feature model is only the representation of family characteristics, variability relations and the configuration space. Accordingly, 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 the development of families of method-oriented architectures.

In order to clarify the difference between a feature and a method service, let us consider the process of method construction as a process of problem solving, where 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 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 problem space, and a feature model represents a hierarchical representation of the problem space including variability. For instance, the problem space (feature model) of a described family method in our case study at the highest level is decomposed into management, requirement engineering, development, and deployment sub-problems, i.e., features (see Sec.

6). On the other hand, method services form the solution space, in which one or more method services (e.g., sub-solutions), implement (i.e., solve) one or more features (i.e., 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. Figure 4 shows the use case modeling feature (one of the features of the feature model given in Section 6) and its 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.

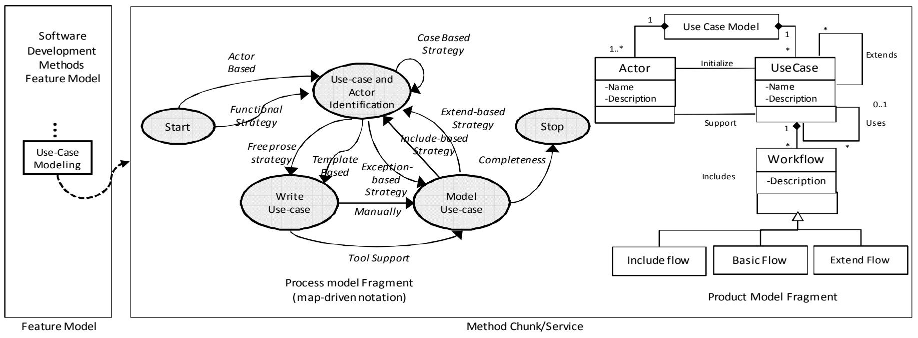


Figure 4: The relation between use case modeling feature (on the left part) and its corresponding method service which define both process and product model for use-case (adapted from [26]).

One of the key concerns in method family engineering is the identification of the right method services for each feature. Having utilized service-oriented standards for defining method services, existing service discovery approaches can be employed to find proper method services for the features. However, service discovery techniques in SOA are limited to syntactical matching between service descriptions and key-words. Recently, applying Semantic Web technologies in the context of SOA has provided benefits for meaningful service publication, discovery, interoperability, and composition. In semantically enabled SOA, several extensions like SAWSDL, OWL-S, and WSMO have been proposed to represents web service descriptions and to annotate them with ontology concepts. Similar to semantically enabled SOA, in order to create a meaningful correspondence between the problem space (i.e., feature model) and the solution space (i.e., method services), we propose to incorporate Semantic Web technologies into both feature models and method services. As shown in Figure 5, the proposed framework encompasses a Semantically Annotated Feature Model Description language (SAFMDL) [20], which aims at describing semantic feature models, a semantic description for method services which is represented by using SAWSDL, and semantic description shared between both spaces which can be seen as the glue for better discovery of method services for features. By using this framework in the context of method-oriented architecture, method engineers can annotate features using the concepts described in the domain ontology. Next, semantic service discovery techniques can be used to find method services which semantically match the features.

After developing the feature model and implementing features by discovering semantic method services in the domain method engineering, in application engineering lifecycle, method engineers select features from feature models corresponding to the requirements of the target method (i.e., a step feature configura-tion). 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 itera-tively until a suitable method is reached for the target problem and is then deployed.

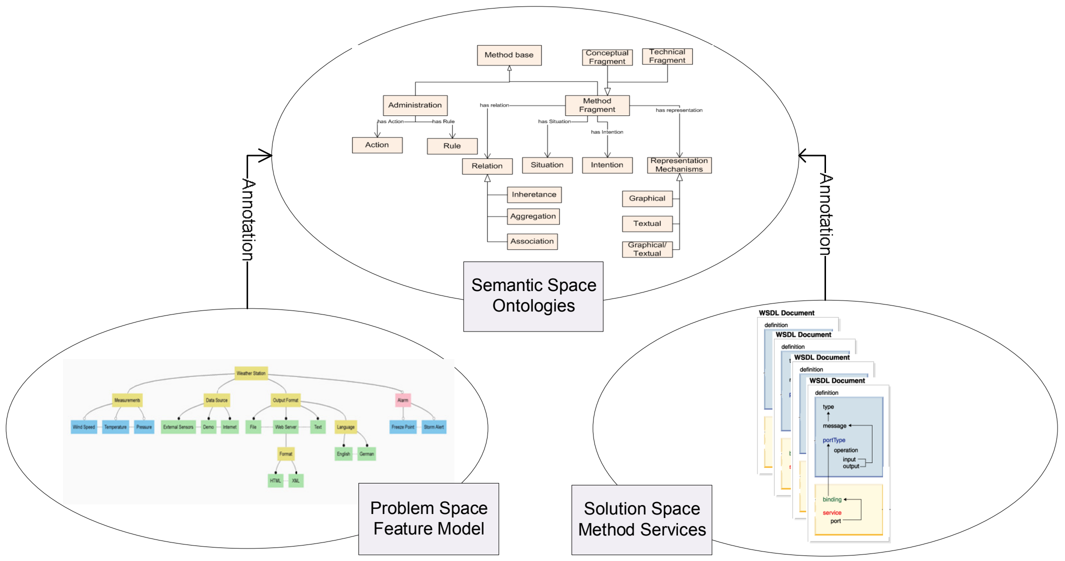


Figure 5: The relation between the problem space (i.e., feature model), solution space (i.e., method services), and semantic space (i.e., domain ontologies) - adapted from [20].

3.3 Semantically-enhanced Feature Models

As shown in figure 5, in order to improve the process of discovering method services for features, we proposed the use of semantic technologies. There are modeling languages which can be used for semantically defining method services (i.e. SAWSDL), however, no semantic representation of feature models exist. On the other hand, a semantic representation of feature models which enables annotation of features with semantic meaning is required to improve the discovery process. Hence, we proposed the Semantic Annotation Feature Model Description Language (SAFMDL) [20]. SAFMDL is a representation of the feature model using the Web Ontology Language (OWL) and it provides proper concepts and relationships that are available in the feature modeling meta-model. In SAFMDL, the structure of feature model is based on two main ontology concepts: *root* and *feature*. Features are refined into *optional* and *mandatory* concepts. Relations between features are modeled through object properties. The possible properties between features concepts are *Sibling relationship*, *AND, OR, XOR, require, exclude,* and *parenthood* (for more details, interested readers are referred to [20]).

In addition to proper OWL classes and properties, SAFMDL extends feature model representation by introducing four new properties for features that reference concepts within external shared ontologies. The additional properties are:

- *selfModelReference* refers to the concepts or notions of domain which the feature represents;
- *preConditionModelReference* refers to the concepts and notions that the feature relies to be realized;
- *postConditionModelReference* represents the concepts of external ontologies that can be realized by the feature;
- *NonFunctionalModelReference* refers to the concepts of non-functional characteristics, which are satisfied by the feature.

These properties defined within SAFMDL are declared as the object properties, which enable designers to add meaning to features through adding concepts from domain ontologies. Figure 6 shows a SAFMDL representation of the *improve role* feature, as a part of the family method given in our case-study, which refers to the activity for improving the modeling of associations between two classes by partitioning classes based on a declared association relationship. The input and output for this feature are UML class diagrams. As depicted in Figure 6, the *improve role* feature is grounded/annotated using the *software design* concept within the *SPLEvaluation* ontology (characteristic ontology in our case-study) which can be considered a common ontology within an organization or agreed collaboration between organizations. Moreover, the *PreconditionModelReference* and *PostconditionModelReference* properties are annotated with values of the URI of the concepts that define the class diagram in UML ontology. Finally, this feature is annotated with the *CMMIOptimizing* concept from the *SPLEvaluation* ontology, which indicates that method services implementing this feature must be aligned with optimizing level of CMMI.

```
<fmdl: Optional rdf:ID= “ImproveRole”>
    <safmdl:selfModelReference  rdf:resource=”http://example.org/ontology/Characteristics/SPLEvaluation.owl#SoftwareDesign”
    <safmdl:preConditionModelReference rdf:resource=”http://example.org/ontology/UML/UMLOnto#ClassDiagram”/>
    <safmdl:postConditionModelReference rdf:resource=”http://example.org/ontology/UML/UMLOnto#ClassDiagram”/>
    <safmdl:NonFunctionalModelReference rdf:resource=”http://example.org/ontology/charactrestics/SPLEvaluation.owl#CMMIOptimizing”/>
    <fmdl:or rdf:resource=”#ImproveAssociation”>
    <fmdl:and rdf:resource=”#designpatterns”>
</fmdl:optional>
```

Figure 6: SAFMDL representation of *improve role* feature.

Having annotated features with concepts from the domain ontology (e.g., in our case-study *SPLEvaluation* ontology), we can apply semantic search mechanisms to identify method services implementing features.

3.4 Method Services Discovery for Features

Having enriched features and method services with semantic information, it is easier to identify method services that syntactically and semantically match the features. We need to identify proper language that provides us with the capability of querying both features defined by SAFMDL and method services defined by SAWSDL. Since SAFMDL models conform to OWL, we are able to use SPARQL queries [22], the W3C candidate recommendation query language for RDF, to manipulate feature models and their constituting features. However, we cannot apply SPARQL for querying SAWSDL documents because SAWSDL is an XML format for representing semantic web service information. On the other hand, XQuery language [23] is a suitable language for querying SAWSDL documents, but it is not suitable for OWL statements.

The XSPARQL [24] W3C submission has been acknowledged as a new language that results from the merge of XQuery and SPARQL, which can deal with both types of XML and OWL data formats. Hence, it provides mapping between data represented by XML and triple formats (e.g., RDF or OWL). Therefore, it is an ideal query language to find the right correspondences for the features of a feature model defined as an SAFMDL instance in the set of available method services expressed through SAWSDL. Furthermore, since many ontologies are expressed by either using OWL or some extensions of the RDF triple format, it is also possible to query the sources of the semantic annotation information that have been used to describe both the feature models and the Web services. In other words, to identify proper method services for features of a method family, three source of information must be integrated. These sources of information are: 1) semantically annotated feature models; 2) semantically annotated method services; and 3) the sources of the semantic information, i.e., domain ontologies. These three sources of information are either expressed in a valid XML-based format or through some extensions of the RDF triple-based format. Thereby, appropriate XSPARQL queries can consolidate these sources of information and enact the realization of problem space models using semantic Web services.

Figure 7 shows an XSPARQL query example written for the *improve role* feature. First, the value of the self-model reference annotation property is acquired. Next concepts from the reference ontology (i.e., SPLEvaluation ontology), as the further specialization of that concept, are extracted. Finally, the set of method services that are annotated with ontological concepts are identified using a suitable query. The output of this query is a set of method services whose semantic information match the semantic information of the *improve role* feature.

```
declare namespace xs   = "http://www.w3.org/2001/XMLSchema";
declare namespace rdfs = "http://www.w3.org/2000/01/rdf-schema#";
declare namespace fmdl = "http://ebagheri.athabascau.ca/spl/fmdl.owl#";
declare namespace safmdl = "http://ebagheri.athabascau.ca/spl/safmdl.owl#";
declare namespace rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#";
declare namespace fm = "http://www.example.org/MOAexample/FamilyMethod.owl#";
declare namespace wsdl = "http://www.w3.org/ns/wsdl";
declare namespace sawsdl = "http://www.w3.org/ns/sawsdl";


for $y from <http://ebagheri.athabascau.ca/temp/gpl.owl>
     where {
                              fm:improverole safmdl:selfModelReference  $y.
                             }

                       construct
                                  {
                                    {
                                     for $x from  <http://example.org/ontology/charactrestics/SPLEvaluation.owl>
                                                      where {
                                                             $x rdfs:subClassOf $y.
                                                            }

                                  construct
                                  {
                                     {

                                         let $doc := doc ("http://example.org/temp/improverole.sawsdl")

                                         let $operations := $doc//wsdl:operation

                                         for $o in $operations

                                          let $el := if ($o/@sawsdl:modelReference=$x) then $o/@name else "Not Suitable"

                                            construct

                                                   {

                                                     [ :suitableService $el; ].

                                                    }


                                         }
                                                    }
                                  }
                         }
```

Figure 7: A sample XSPRQL query for the improve role feature.

4 Families of Method-Oriented Architecture

Similar to developing software product lines, we propose two main lifecycles for the 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 variabilities are modeled using a feature model, and suitable method services corresponding to the 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, which results in assembled method services corresponding to the configured feature model. The method application engineering lifecycle is carried out every time when 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.

4.1 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. Figure 8 illustrates the phases and stages of the method domain engineering lifecycle.

Method domain engineering starts with the *Method Family Scoping* phase which aims at determining a set of software development methods belonging to the family. In this phase, first using expert knowledge 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 are derived. It systematizes the method product information, identifies the main features of the product line and checks the consistency. With regard to the characteristics 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. Next, according to the preconditions established in the previous stage, the proper domain ontologies are identified (developed) and precise functionality of the method components that should be supported by the method family are described. Domain ontologies include ontologies containing concepts related to products, processes, and situational characteristics. Process and product ontologies define process concepts (like phase, stage, and task) and product concepts (e.g., class diagram, class, and requirements), respectively. Characteristics ontology includes concepts related to the application domain, method types, development approaches, and quality standards (e.g., CMMI and SPICE). An example of the characteristics ontology is shown in Figure 10.

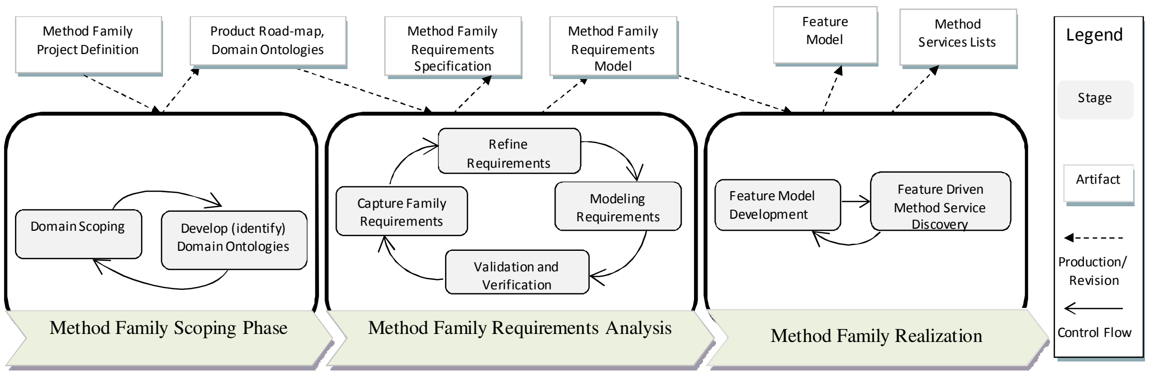

Method Family Project Definition
Product Road-map, Domain Ontologies
Method Family Requirements Specification
Method Family Requirements Model
Feature Model
Method Services Lists
Legend
Stage
Artifact
Production/ Revision
Control Flow
Domain Scoping
Develop (identify) Domain Ontologies
Refine Requirements
Capture Family Requirements
Modeling Requirements
Validation and Verification
Feature Model Development
Feature Driven Method Service Discovery
Method Family Scoping Phase
Method Family Requirements Analysis
Method Family Realization

Figure 8: The Method Domain Engineering Lifecycle

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 of each requirement as well as the variability which exists within requirements. The phase receives the documents, stakeholders' viewpoints, and the product-line roadmap. Similar to typical software engineering procedures, we define *functional requirements* and *non-functional requirements* for methods. The functional requirements determine 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 receives the 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 [[14]] are applied to develop the family requirements model. The family requirements model includes the functional requirements and is represented a 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 [[14]], 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 individual 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, the method engineer annotates features using concepts from the domain ontologies, e.g., process, product, and situational characteristics ontologies.

A Feature-Driven Method for service discovery and selection follows feature modeling step for a method family. The stage of the feature-driven discovery is performed by transforming feature model and its annotation into SAFMDL representation and producing XSPARQL queries for each feature based annotations. In essence, feature annotation provides functional and non-functional concepts 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 XSPARQL queries for discovering semantic method services, which are defined by SAWSDL. Given that there are no publicly available repositories of method services, we are now developing a test collection of method services.

4.2 Method Application Engineering

Once the method domain model is created, the method engineers can take the method domain model and create different instances of it based on target method requirements. Hereafter, we refer to this process as *method application engineering*.

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 (i.e., method services) 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 Figure 9.

The *Application Method Requirement Analysis* phase aims at defining 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) [15]. The family requirements model and documents are utilized through this phase for producing method application requirements.

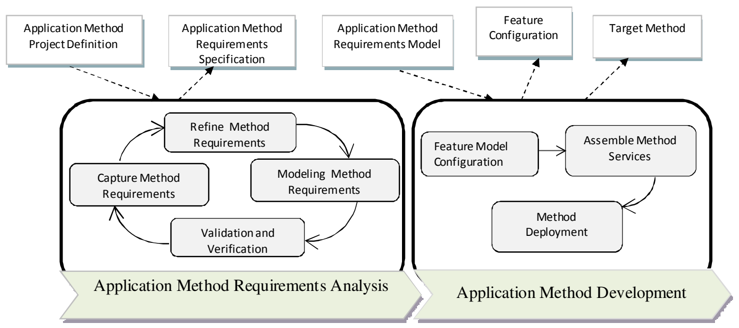

Application Method Project Definition
Application Method Requirements Specification
Application Method Requirements Model
Feature Configuration
Target Method
Refine Method Requirements
Capture Method Requirements
Modeling Method Requirements
Validation and Verification
Feature Model Configuration
Assemble Method Services
Method Deployment
Application Method Requirements Analysis
Application Method Development

Figure 9: The 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, situational characteristics for the current method are determined by utilizing the characteristics ontology developed through the method domain engineering lifecycle. 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 also 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 the *stage configuration* process [16]. It receives the method requirements and produces the corresponding feature configuration. The staged configuration process 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 non-functional properties defined in the descriptions of method services, the final method service for each feature is selected from the list of alternative method services. The output of this stage is the set of 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 could be discovered through local or distributed repositories otherwise they should be developed from scratch. Once the method engineer makes sure that all the required method services (features) have been aggregated, he/she starts the composition of method services (features) via the *Assemble Method Services* stage. The selected features are related to both functional (e.g., requirement elicitation, use case modeling, and developing design model) and non-functional aspects of the method (e.g. quality assurance, project monitoring, and traceability checking). First, the method services related to functional aspects of method are orchestrated and the necessary adaptation and mediation are conducted. Afterwards, the method services related to non-functional aspects (like quality assurance and project monitoring) are added into proper parts of the method. 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 of the 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's environment by preparing method documents, training developers, and supporting staff through the execution of the method.

5 Case Study

In order to illustrate the challenges and demonstrate the proposed approach, we present a case study that demonstrates an organization, which develops software systems in two distinct domains, namely, information systems (both desktop and web-based systems) and real-time systems. Here, our method fragments are defined using SAWSDL and annotated with proper domain ontologies. In the following, we explain the domain method engineering process for the case study. For each phase of domain method engineering, the steps and produced artefacts are explained.

5.1 Case-Study Description

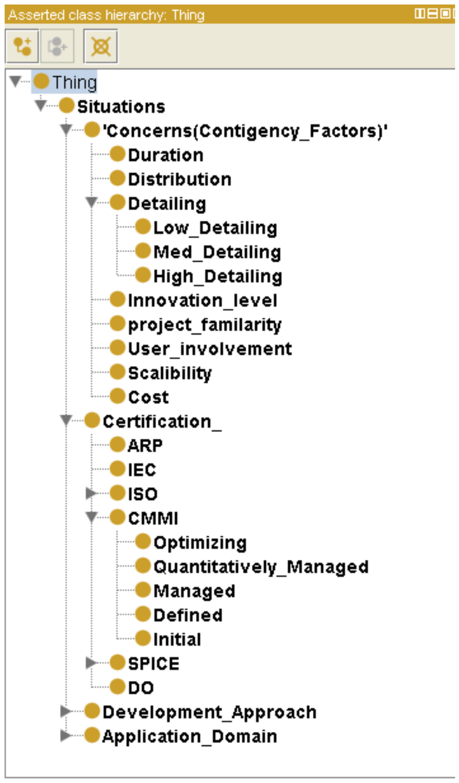
We consider that the organization has adopted a base method (e.g., RUP) for the entire systems development process. The base method is supposedly a modular method and its method components are stored in a method repository. Moreover, the organization has employed different development approaches, including *code centric development, component based development*, and *model driven development*. Based on the scale and complexity of a given 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 the 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 by 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 the 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 3) create a systematic method for adaptation of the base method considering the needs and requirements of the new development situations.

5.2 Method Family Scoping

According to the description of the required method family given in the Section 5.1, we identified the criteria, which specify the product line boundaries, and the main functionality area. The main criteria for boundaries of product lines are *technical domains* (i.e., application properties), *development approaches, and contingency factors.* The major activities supported by the family method are determined based on the needs for coverage of generic software development lifecycle and project management. Table 1 shows a part of the product line scoping results. Since we assumed the design and analysis are documented using UML, we developed an ontology with concepts corresponding to elements in the UML meta-model. For example, in our product ontology we have concept class diagram that corresponds to class diagram in the UML meta-model. Later on, these concepts are used for annotating features in the feature model showing the artefacts required and generated by the features. Moreover, the characteristics ontology defines concepts related to application properties, development approach, contingency factors, and quality standards. As shown in Figure 10, *Certification* is a concept in the characteristics ontology that is specialized in *CMMI, DO, IEC, ISO,* and *SPICE* concepts. The CMMI concept is further specialized into five concepts showing different level of CMMI.

Table 1: the result of method family scoping phase

Stage	Prod-	Descriptions
Domain Scoping	Criteria for Boundaries	• *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.
	Major Activities of the family	• 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). • *Project Management* – monitoring, risk management, configuration and change management, postmortem reviewing, metric management, human resource management.
Develop	Ontologies	• Process Ontology, Product Ontology (UML concepts), Characteristics ontology (SPL evaluation Ontology).



- Thing
  - Situations
    - 'Concerns(Contigency_Factors)'
      - Duration
      - Distribution
      - Detailing
        - Low_Detailing
        - Med_Detailing
        - High_Detailing
      - Innovation_level
      - project_familarity
      - User_involvement
      - Scalibility
      - Cost
    - Certification_
      - ARP
      - IEC
      - ISO
      - CMMI
        - Optimizing
        - Quantitatively_Managed
        - Managed
        - Defined
        - Initial
      - SPICE
      - DO
    - Development_Approach
    - Application_Domain

Figure 10: Characteristics ontology for the given case-study

5.3 Family Requirement Analysis

In this phase, first functional and non-functional requirements and their commonalities and variabilities are captured and documented, separately. Table 2 shows a part of the requirements categorized based on their types. Functional requirements include the activities and work products that should be supported by a family of methods. The family requirements model is created first by using the map-driven approach [14] and subsequently verified and validated. Figure 11 shows a part of the map model for the family method. In the figure, four intentions are defined for the family method including *elicit requirements, documents requirements, analyze requirements*, and *manage requirements*. In order to achieve the *elicit requirements* intention, we can follow one of *survey, goal identification, workshop,* and *interview* strategies. Moreover, requirements can be documented via *use-case, activity diagram, VFM,* or *user stories*.

Characteristics ontology is extended with new concepts identified based on non-functional requirements. For example, concepts related to *detailing* and proper refinement levels (i.e., *Low, medium,* and *high*) are added into the characteristic ontology.

Table 2: part of requirements for method family

Functional and non-functional Requirements	
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.
Non-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.

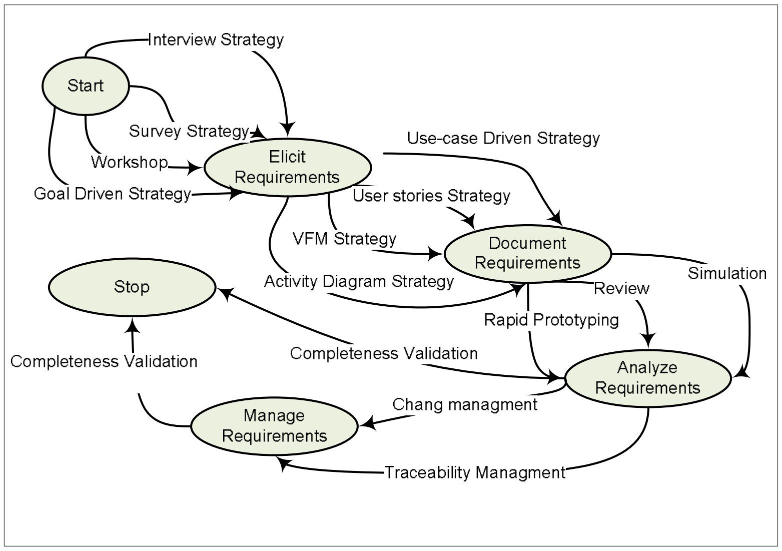


Figure 11: Requirements model for requirements engineering phase of method family.

5.4 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 Figure 12. The features show the method services required for the family and they can be considered as interfaces for representing method services in the family. The main features in the feature model include *management, requirements engineering, development*, and *deployment*. These features are further refined and decomposed into more fine-grained features, and their variabilities are represented using variability relations within the feature model. Features are annotated with concepts from the characteristics ontology. For instance, the feature *use-cases* modeling is annotated with *medium detailing,* and *medium cost* concepts from the characteristics ontology.

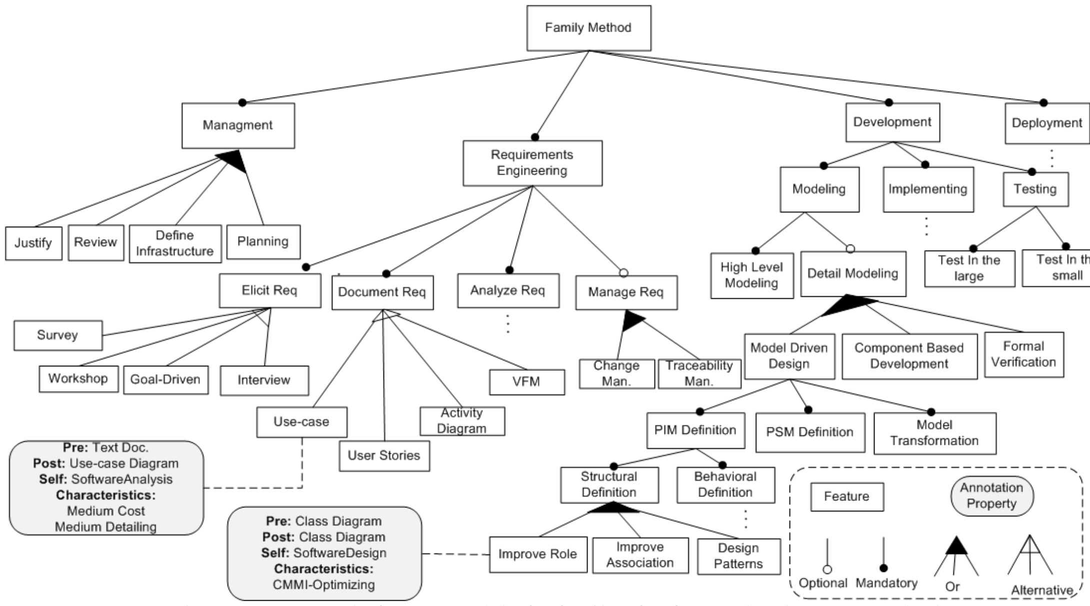


Figure 12: A sample feature model of a family of software development methods

5.5 Feature-driven Method Service Discovery and Selection

Feature model development is followed by the process of discovery of method services. This process aims at finding and selecting among available methods services, which can match and satisfy desired functional and non-functional requirements of the method for the specific situations. As we described earlier, we transform each feature and their associated annotations into SAFMDL and generate XSPARQL queries for discovering method services stored in method repositories. Therefore, our assumption is that the method components, which are described by SAWSDL, are exposed as services and are available and accessible through either the proprietary method repositories of the organization or public repositories provided by third parties. Therefore, organizations can develop their internal repository of method services and use the framework for discovering local method services or find method services in online method repositories like Open Process Framework (OPF) [30]. Although there are on-line repositories such as Open Process Framework (OPF) [30], available reusable method components are not accessible through standard interfaces. Moreover, there are no facilities to search and discover such available and existing methods. We are developing a small test set of method services in a local repository to perform feature driven method service discovery.

6 Discussion and Tooling Support

Two main issues regarding the present proposal are its validity and cost-benefit analysis. For both issues, we need to conduct an empirical study. We have reported 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 and address these issues, we provide analytical discussion based upon the analogy between software and methods as proposed by Osterweil [33]. In accordance, our assumption is “software processes are software too” [33]. Considering this analogy, we can adopt similar approaches and techniques used in software engineering for solving existing problems in method engineering. As it can be observed, the method engineering community proposed MOA being inspired by SOA to deal with the lack of standard for defining well-defined interfaces for method components, exposing them as services and taking the advantages of SOA [1]. Nevertheless, to use this 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 take the benefits of SOA in the MOA domain. The other problem that method engineers deal with is that variability may exist in the base method [6][13]. On the other hand, software variability is a well-known problem which has been studied by the software product line engineering community and numerous well-established techniques and approaches have been proposed like feature modeling to address the problem, and various success stories in using product families and associated techniques have been reported from industry. 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 [34]. These results ensure both validity and benefits of software families. Therefore, we tried to make an analogy between software families and method base and coined the notion of *family of methods*. We mapped the features to method component 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 from this in 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 compensate these costs as happened in the broader software engineering practice.

In order to support method engineers in modeling and configuring method family, we are implementing a chain of tooling supports called *Product Line Composer (PLC)*. PLC extends Feature Model Plug-in (*fmp*[1]), available for Eclipse environment, and enables the annotation of features with non-functional concepts (characteristics ontology) and other domain ontologies. Additionally, a set of transformation rules for generating SAFMDL, and a view for ontological representation of feature models is provided in PLC environment. For the next stage, we are working to incorporate the XSPARQL language with our tooling support for formulating and executing queries on the repository of semantic method services. Additionally, in our tool a set of decision making algorithms like Stratified Analytical Hierarchy Process (S-AHP) [25] are implemented to help method engineers during the specialization of the method family.

7 Related Work

Method engineering aims at providing techniques and approaches for constructing and/or adapting 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 is 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 nents [7][26][35]). These approaches mostly focus on reusability and modularity principles. Besides this classification, Karlsson et al. [13] proposed the *Method Configuration* approach (more general than extension-based) where a target method is created by adding/removing elements and features. These authors concentrate on variability management and reusability. All mentioned approaches are established based

[1] http://gsd.uwaterloo.ca/projects/fmp-plugin/

upon one or more of the following principles: meta-modeling, reuse, modularity, and flexibility. Our proposed approach is similar to assembly-based approaches and method configuration by following modularity, reusability, and variability principles. However, our approach enables a higher degree of reusability by leveraging SOA principles and catering systematic variability management by employing SPLE principles (as it is proven and shown in software engineering that SPLE increases reusability [34]).

Gonzalez-Perez [31] 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 [32] enriched the ISO/IEC 24744 for creating and tailoring methods through an approach called Application-based Domain Modeling (ADOM). The approach is based on a 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 situational method components and situational methods, which are created based on the 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 template-based approach where reference architecture for the entire family is designed and eases the development and assembly of methods. Additionally, we use a new concept for the method component (i.e., method service), which utilizes standards in SOA to improve discovering and reusing method components.

Similar to our work, Kornyshova et al. [36] proposed the notion of method family to organize method components of a specific domain. They categorized method components into mandatory and optional where mandatory method components must be selected in the final method and optional may or may not be selected. They defined three processes for developing method family namely method family definition process, method line configuration process, and method line application process. We applied a well-established variability modeling language (i.e., feature model) and employed method services instead of method components.

Recently MOA [18][1] was proposed which aims at empowering and enhancing the assembly-based method engineering principles with a standard for describing method components (in terms of method service) and taking the advantages of SOA for addressing the discovery and retrieving of existing distributed method components. Our approach also utilizes MOA to describe and discover method services corresponding to the features of a method family. Moreover, we incorporated semantics into method services, which improves the identification of proper method services.

Application of ontologies and semantic web technologies in method engineering has been investigated by some researchers in the domain of method engineering [38][39][40]. Niknafs et al. [38] proposed an ontology which define the main concepts and relations required for a method base. The main concepts in the proposed ontology are method fragment and administration. Mirbel [40] proposed a lightweight top ontology which define the core concepts required for qualifying knowledge about method. Iacovelli and Souveyet [39] developed an ontology of method descriptor as a domain ontology for method engineering and investigated five existing method descriptors using the concepts of their ontology. These works show the interest of method engineering community to develop common ontologies. These ontologies can be used both for annotating method services and features in our approach to add meaning to these elements in our context.

8 Conclusion and Future Work

Existing method engineering approaches have addressed several challenges in developing and configuring methods for information systems. However, according to [37], still there are many research challenges which need to be addressed. In this paper, we targeted two main challenges including the need for techniques to discover proper method services for the current situation and the need for managing variability in base method. Hence, by utilizing service oriented product line and semantic web technologies, an approach for developing families of modular software developments methods was proposed. In our approach, we enhanced the notion of method service by adding semantic meaning and knowledge of domain (i.e. ontologies) into description of method services. This improvement addresses the first challenge via utilizing an existing standard semantic enabled query language (i.e. XSPRQL) for identifying method services that matches with requirements both syntactically and semantically. In order to manage variability of method families, the proposed approach makes use of feature modeling, the most well-known variability modeling language in software product line engineering. Feature models are employed to represent the variability in a method family in terms of difference in its activities and situational characteristics. This aspect of our proposal addresses the second targeted challenge. Additionally, in order to address both challenges at the same time, i.e. representing variability in the method family and identifying proper semantic method services, a proposed framework use semantic web technologies (i.e. ontologies) to bridge problem space (feature model) and solution space (method services). Hence, semantic representation is defined for features and a standard query language is used.

We believe that the described concept of families of semantically enabled method-oriented architectures is feasible within an organization with well-established domain knowledge formulated in domain ontologies and repository of method services. Although it requires effort for annotating method services and features by semantic meaning, but this effort is only needed one time. Afterward, every time a new method is required, the method can easily be derived from the method family by performing steps proposed in method application engineering.

As a future work, we aim at completing our tool (PLC) and add decision oriented algorithms to help method engineers in deriving situational methods in semi-automatic way. Moreover, we are implementing generation of XSPARQL queries from features and their annotation properties. In addition to tooling aspect, we intend to provide a more comprehensive evaluation of the proposed approach by developing a collection of semantic method services to be used for experimentation.

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] Martınez-Ruiz, T., Garcıa, F., Piattini, M., Muench, J.: Modelling software process variability: an empirical study. Software, IET 5(2) (April 2011) 172–187

[8] Simidchieva, B.I., Clarke, L.A., Osterweil, L.J.: Representing process variation with a process family. In: Proceedings of ICSP'07, Berlin, Springer (2007) 109–120

[9] J. Kopecky, T. Vitvar, C. Bournez, and J. Farrell. 2007. SAWSDL: Semantic Annota-tions for WSDL and XML Schema. IEEE In-ternet Computing 11(6): 60-67.

[10] D. Martin, M. Burstein, D. Mcdermott, S. Mcilraith, M. Paolucci, K. Sycara, D. L. Mcguinness, E. Sirin, and N. Srinivasan. 2007. Bringing Semantics to Web Services with OWL-S. World Wide Web 10(3): 243-277.

[11] R. Lara, D. Roman, Axel Polleres, and Dieter Fensel. A Conceptual Comparison of WSMO and OWL-S. In ECOWS 2004, volume 3250 of LNCS, pages 254–269. Springer, 2004.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] Klusch, M.: Semantic Web Service Coordination. CASCOM: Intelligent Service Coordination in the Semantic Web, 2008, pp. 59-104.

[20] Bagheri, E., Asadi, M. , Ensan, F., Gasevic, D., Mohabbati, B., " Bringing Semantics to Feature Models with SAFMDL," Proceedings of the 2011 Conference of the Centre for Advanced Studies on Collaborative Research (CASCON 2011), Markham, Canada, 2011.

[21] 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.

[22] E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF. W3C Candidate Rec. 6 April 2006. http://www.w3.org/TR/rdf-sparql-query/.

[23] D., Chamberlin, D. Florescu, J. Robie, J. Simeon, and M. Stefanescu, (2001). XQuery: A Query Language for XML. Working draft, World Wide Web Consortium. See http://www.w3.org/TR/xquery/.

[24] W. Akhtar, J. Kopecky, T. Krennwallner, and A. Polleres. 2008. XSPARQL: traveling be-tween the XML and RDF worlds - and avoid-ing the XSLT pilgrimage. In Proceedings of the 5th European semantic web conference on The semantic web: research and applica-tions (ESWC'08).

[25] Bagheri, E., Asadi, M., Gašević, D., Soltani, S., "Stratified Analytic Hierarchy Process: Prioritization and Selection of Software Features," *In Proceedings of the 14th International Software Product Lines Conference*, Jeju Island, South Korea, 2010, pp. 300-315.

[26] Mirbel, I., Ralyte J.: Situational method engineering: combining assembly-based and roadmap-driven approaches. Requirements Engineering 11(1): 58–78, 2006.

[27] 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.

[28] 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.

[29] Bošković et al. (2010)Automated Staged Configuration with Semantic Web Technologies, International Journal of Software Engineering and Knowledge Engineering (in press)

[30] OPEN Process Framework (OPF) Web Site, http://www.opfro.org/

[31] 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.

[32] 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.

[33] Osterweil, L.: Software processes are software too. Proceedings of the ICSE. pp. 2-13IEEE Computer Society Press, Monterey, California, United States (1987).

[34] 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).

[35] 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.

[36] Kornyshova, E., Deneckère, R., Rolland, C.: Method Families Concept: Application to Decision-Making Methods. In: Halpin, T., Nurcan, S., Krogstie, J., Soffer, P., Proper, E., Schmidt, R., and Bider, I. (eds.) Enterprise, Business-Process and Information Systems Modeling. pp. 413-427. Springer Berlin Heidelberg, Berlin, Heidelberg (2011).

[37] Henderson-sellers, B., Ralyté, J.: Situational Method Engineering: State-of-the-Art Review, J. UCS 16(3): 424-478 (2010)

[38] Niknafs, A., Asadi, M., Abolhassani, H.: Ontology-Based Method Engineering. International Journal of Computer Science and Network Security. IJCSNS 7(8) (2007)

[39] Iacovelli, A., Souveyet, C.: Towards Common Ground in SME: An Ontology of Method Descriptors. In: Ralyté, J., Mirbel, I., and Deneckère, R. (eds.) Engineering Methods in the Service-Oriented Context. pp. 77-90. Springer Berlin Heidelberg, Berlin, Heidelberg (2011).

[40] Mirbel, I.: Connecting Method Engineering Knowedge: a Community Based Approach. In: Proceedings of ME 2007, Geneva, Switzerland (2007)