Leveraging CVL to Manage Variability in Software Process Lines

Emmanuelle Rouillé, Benoit Combemale, Olivier Barais, David Touzet, Jean-Marc Jézéquel

Leveraging CVL to Manage Variability in Software Process Lines

Emmanuelle Rouillé[*†], Benoît Combemale[†], Olivier Barais[†], David Touzet[*] and Jean-Marc Jézéquel[†]
[*]Sodifrance
P.A. la Bretèche, avenue Saint-Vincent, 35768, Saint-Grégoire, France
Email: {erouille, dtouzet}@sodifrance.fr
[†]Université de Rennes 1
IRISA, Campus de Beaulieu, 35042, Rennes, France
Email: {emmanuelle.rouille, benoit.combemale, olivier.barais, jean-marc.jezequel}@irisa.fr

Abstract—**Variability on project requirements often implies variability on software processes. To manage such variability, Software Process Lines (SPLs) can be used to represent commonality (i.e., common practices) and variability (i.e., differences) of a set of related software processes. To this end, some Software Process Modeling Languages (SPMLs) natively integrate variability mechanisms. Nevertheless, such a coupling between the SPML and the variability mechanisms i) requires to interpret the requirements variability in terms of the processes variability, ii) limits the reuse of the requirements variability for other purposes (*e.g.,* the development itself), and iii) is a barrier to the use of advances from the field of variability management. In this paper, we propose an approach to apply the Common Variability Language (CVL from the OMG consortium) for requirement variability modeling and its binding to the processes. This work is illustrated on a family of industrial Java development processes. Our approach enables the definition of an SPL and the automatic derivation of a process from this SPL according to the requirements of a given project. The variability is managed separately from the process model and benefits from existing tools coming from process modeling community and CVL.**

I. Introduction

A software development process captures the sequence of steps to perform in order to realize a software engineering project. Processes are a mean for capturing the know-how that companies acquired during projects. Inherent variability on project requirements often implies variability on software processes that are still complex to capture. Moreover, the more variants of a software artifact there are, the more the management of variability reduces development costs and time to market [1]. This is also true in context of processes. Some approaches for modeling process variability and for enabling the reuse of process model rely on Software Process Line Engineering (SPLE) [2]. Indeed, SPLE enables the definition and reuse of common and variable parts between processes. These approaches also use the Model-Driven Engineering (MDE) in order to define a Software Process Line (SPL), which is a set of processes that captures commonalities and variabilities between these processes [3]. However, existing contributions are dependent of a process metamodel. Indeed, some of them modify a process metamodel with a variability concern, in order to model places in processes that vary and the possible variations. This prevents the reuse of the variability concern with other process metamodels without adapting it. This also requires the development of new tools (e.g. tools for process modeling, execution, checking) for managing the modified process metamodel. Other contributions do not modify the process metamodel but transform the process model into a pivot structure that captures the variability concern. These contributions are also dependent of the process metamodel because they require the definition of a transformation for each process metamodel. We still miss an approach for automatically reusing processes according to projects requirements that is independent of the process metamodel.

As stated in the specification of the Common Variability Language (CVL)[1], CVL is a "domain-independent language for specifying and resolving variability". Therefore, CVL would address this limitation. Indeed, CVL is an OMG[2] standardization effort for the product lines definition. It enables to reference process elements defined in every process model whose metamodel conforms to the Meta-Object Facility (MOF)[3] metametamodel and it provides operators to specify how to derive a process.

In this article we answer the following research questions:

1) Rq1: how to use CVL in the context of processes?
2) Rq2: does CVL enable the management of processes variability?
3) Rq3: is CVL independent of the process metamodel?

To answer these research questions, we lead an experiment with Sodifrance[4], a software and computing services company. Sodifrance performs different kinds of projects that are development, modernization and software maintenance projects. This experiment consists in modeling and/or refactoring existing software development processes to enable their reuse according to commonalities in projects requirements. For this experiment, we use the Software Process Engineering Metamodel (SPEM) 2.0 [4] as process metamodel. Indeed, SPEM 2.0 is an OMG standard and commercial modelers exist. However, we could choose any other process metamodel

[1]http://www.omgwiki.org/variability/
[2]http://www.omg.org/
[3]http://www.omg.org/mof/
[4]http://www.sodifrance.fr/

according to one's needs and preferences. We perform this experiment and illustrate our approach on a process line of Java development processes.

This paper is organized as follows. Section II presents our illustrative example. Section III presents SPEM 2.0 and CVL. Sections IV, V and VI respectively answer the research questions rq1, rq2 and rq3. Section VII presents the related work. We conclude in Section VIII.

II. Illustrative Example

In this section, we present a simplified Java development process of the Sodifrance company as an illustrative example.

A. The Most Often Used Java Development Process

The most often used Java development process of the Sodifrance company is as follows. During the first task, the customer produces a specifications document defining the technical and functional specifications of the application to develop, according to the project's requirements. The following task, the development, consists of manually writing the Java code of the application. It takes as input the specifications document and produces as output the Java code. During the tests task, the developers test the application under development. If there are errors, then the process flow goes back to the development task in order to correct the errors. If there is no error, then the process flow goes to the production task, which consists of deploying the application on the customer environment. During the development task, the developers have to use the Eclipse IDE and a version control system, which is SVN in this example. They also use MySQL as database during the development task. Modelling the process permits to create the initial configuration of a new project: building eclipse IDE, generating the maven project object models, initializing the source code repository, ...

B. Variability in the Java Development Process

There are variants of the Java development process. For instance, if the customer uses Oracle as database, then developers will use Oracle instead of MySQL. If there is no data to persist, then there is no database. If the customer provides the specifications into a formal language (*e.g.,* UML 2.x), then a code generation task occurs before the development task. It consists of generating the code which is similar for all the entities of the application. It produces some generated Java code as output. When the code generation task occurs, then the development task takes the generated Java code as additional input and consists of manually writing the non generated code.

III. Background

In this section, we present SPEM 2.0 and CVL.

A. SPEM 2.0

We use the SPEM 2.0 process metamodel to model the Java development process example. The SPEM 2.0 specification defines a metamodel whose general idea is that roles perform activities, that take work products as inputs and outputs. The peculiarity of SPEM 2.0 is that it separates the definition of the process elements (the method content) from their use into processes (the process structure).

Figure 1 shows an extract of the SPEM 2.0 metamodel we use in the following. However, one can use every SPEM 2.0 concepts with our methodology.

The method content part contains method content elements (*MethodContentElement*) for defining role definitions (*RoleDefinition*), that perform task definitions (*TaskDefinition*), using a tool definition (*ToolDefinition*). A task definition uses work product definitions (*WorkProductDefinition*) as inputs and outputs.

The process structure part contains work breakdown structures (*BreakdownElement* and *WorkBreakdownElement*) and structures for describing workflows (*Activity*, *Process* and *DeliveryProcess*). A delivery process (*DeliveryProcess*) describes a complete project lifecycle. An activity (*Activity*) is a container for breakdown elements (*BreakdownElement*). Subtypes of the class named *MethodContentUse* use subtypes of the class named *MethodContentElement* through corresponding references. A role use (*RoleUse*) performs a task use (*TaskUse*), which uses a work product use (*WorkProductUse*) as input or output. The parameter type (attribute named *parameterType*) of a process parameter (*ProcessParameter*) defines a work product use of a task use.

We use UML 2 [5] activity diagrams to model the flow of task uses because the class named *TaskUse* is a stereotyped UML 2 Action.

Figure 2 shows the concrete syntax we use to represent SPEM 2.0 and activity diagram model elements.

Figure 3 shows the SPEM 2.0 model of the most often used Java development process of our illustrative example.

B. CVL

We use CVL to define an SPL and automatically derive a process according to the project's requirements from this SPL. Figure 4 summarizes CVL and illustrates it in the context of processes. CVL proposes to define a base model, which is an instance of any metamodel conforming to the MOF. The base model is a base process model in our case. Then, CVL provides constructs to capture the variability (the Variability Abstraction Model (VAM)). For instance, the VAM specifies the projects requirements variability. CVL also provides constructs to define the realization of this variability on the base model (the Variability Realization Model (VRM)). The VRM is the original part of CVL. It allows designer to explicit the mapping between the VAM (that can be compared to a feature model) and the assets (the base model). In this mapping, the VRM defines which base model elements are impacted with a specific variation point but also how these base model elements are impacted. CVL provides in the VRM several types of variation point that capture the derivation semantics. The main benefits of this model is the ability to capture the variability requirement in the VAM and its materialization in the base model using the VRM. In our case, the VRM defines a binding between the projects requirements variability and the processes variability. CVL provides constructs to resolve

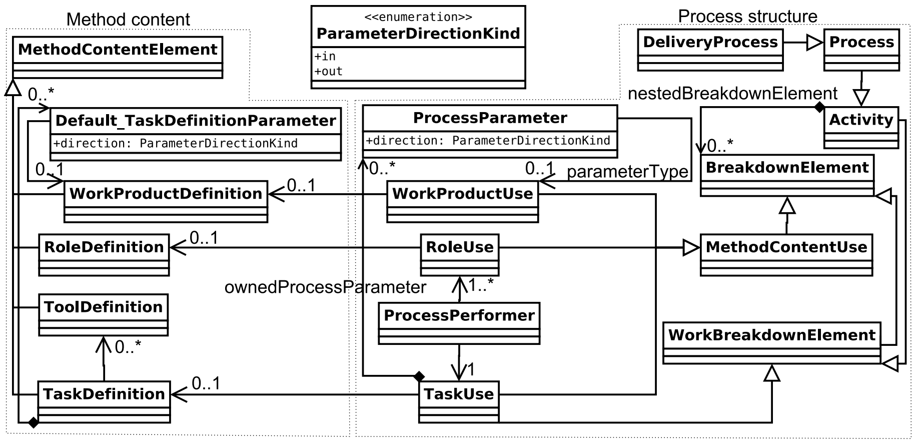


Figure 1: Excerpt of the SPEM 2.0 metamodel

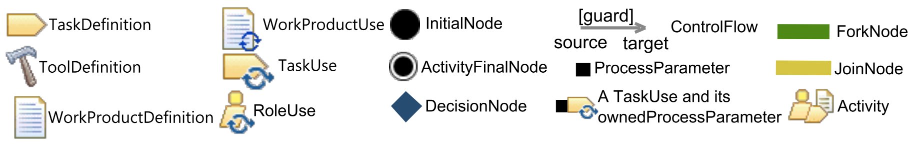


Figure 2: SPEM 2.0 and activity diagram concrete syntax

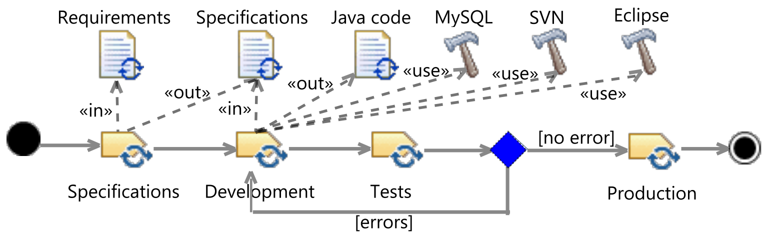


Figure 3: Illustrative example: most often used Java development process

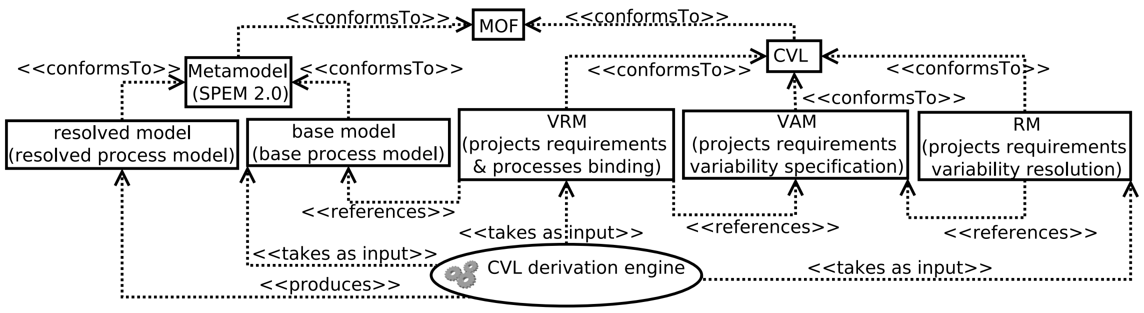


Figure 4: Using CVL for SPLs

this variability in order to select a configuration of the base model (the Resolution Model (RM)).

CVL can be executed. A CVL model composed of a VAM, a VRM and a RM contains enough information to provide a resolved base model without variability. The resolved model is thus another instance of the metamodel that the base model conforms to. In our context, the resolved model is a resolved process model. Thereafter, we present the CVL metamodel according to its three parts (variability abstraction, variability realization and resolution). Figure 5 details the excerpt of the CVL metamodel that we use in the following.

The variability abstraction part defines variability through variability specifications (*VSpec*). A variability specification can be a choice (*Choice*), *i.e.* a feature that will belong or not to the resolved model depending on whether it is resolved to true or false. A variability specification may contain children. Children can be resolved to true only if their parent is resolved to true. A variability specification also has a group multiplicity (*groupMultiplicity*) defining the minimal and maximal (attributes named *lower* and *upper* of the class

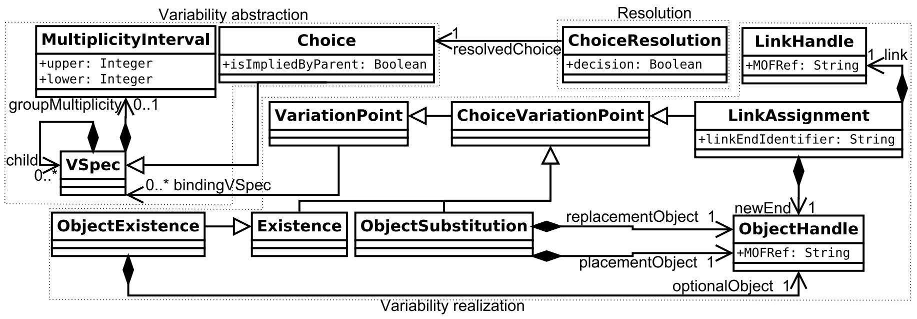


Figure 5: Excerpt of the CVL metamodel

named *MultiplicityInterval*) numbers of direct children that can be resolved to true. If a choice is implied by its parent (attribute named *isImpliedByParent* equals true), then it must be resolved to true if its parent is resolved to true, expressing a mandatory feature. The variability abstraction model provides the same expression power than a feature model. *Xor*, *Card*, *Or*, *Alt*, or *Mandatory* can be expressed using the *groupMultilplicity* reference and the *isImpliedByParent* attribute.

The variability realization part defines the variation points (*VariationPoint*), that are operations to perform on the base model in order to derive a resolved model. A variation point is performed when its binding variability specifications (*bindingVSpecs*) are resolved to true. CVL defines several types of variation points. Each of them can be seen as a reusable function that can be applied on the base model during the derivation. Among variation points, an object substitution (*ObjectSubstitution*) replaces an object of the base model, the placement object (*placementObject*), by another one, the replacement object (*replacementObject*), and deletes the placement object. A link assignment (*LinkAssignment*) sets a reference of the base model, the link end identifier reference, to a new end (*newEnd*). A link object (*link*) contains the reference, identified by its name (*linkEndIdentifier*). An object existence (*ObjectExistence*) specifies that an optional object (*optionalObject*) of the base model will still exist in the resolved model. The optional object is removed from the resolved model if at least one binding variability specification is resolved to false. Thus, object existences execute in negative variability. This means that the objects belong to the base model if they belong to a configuration. They are deleted during derivation if they do not belong to the selected configuration. On the contrary, positive variability means that all the objects are not in the base model and they are created during derivation. The link handle (*LinkHandle*) and the object handle (*ObjectHandle*) model elements reference an object of the base model through their attribute named *MOFRef*, which corresponds to the URI of the object to reference.

The resolution part contains choice resolutions (*ChoiceResolution*) that resolve their resolved choices (*resolvedChoice*) to true or false.

Figure 6 shows the concrete syntax we use to represent the CVL model elements.

We now detail the part of the derivation algorithm which is useful to understand the example of Section IV. The CVL derivation engine performs each variation point whose binding variability specifications are resolved to true. The derivation applies on a copy of the base process model. To perform a link assignment, it retrieves from the base process model copy the link object. If its link end identifier reference has a upper bound of one, it updates the link end identifier reference to the new end. If the link end identifier has an infinite upper bound, it adds the new end to the list of links that instantiate the link end identifier reference. To perform an object substitution, it retrieves from the base process model copy the placement object and applies all its incoming references to the replacement object. It then deletes the placement object from the base process model copy. For an object existence, the derivation engine removes the optional object from the base process model copy if at least one of the binding variability specifications of the object existence is resolved to false. Finally, it serializes the modified copy of the base process model into a new file, giving the resolved process model. Our implementation of the CVL derivation engine relies on the Eclipse Modeling Framework (EMF)[5] API to load, manage and save the models[6].

IV. Approach

In this section, we answer rq1 by presenting our approach to use CVL in the context of processes. Figure 7 shows an overview of this approach. It involves two roles: a process expert, who knows the different processes of a company and their context of use, and an engineer, who is involved into a project and needs a process specific to this project. The process expert captures the requirements variability and its binding to the SPL (steps 1 to 3). Then, the engineer automatically derives a process from this SPL according to the requirements of a given project (steps 4 and 5). Steps 4 and 5 occur each time the engineer wants to derive a process. In our approach, the process expert and the engineer use the CVL tooling to perform steps 2 to 5. Any SPML can be used to perform step 1, even if we focus in this paper on the use of SPEM. Our approach preserves the separation between projects requirements and processes. It also directly binds projects requirements to processes, instead of interpreting requirements variability in terms of processes variability. In

[5]http://www.eclipse.org/modeling/emf/
[6]CVL derivation engine can be downloaded http://goo.gl/ifGFD

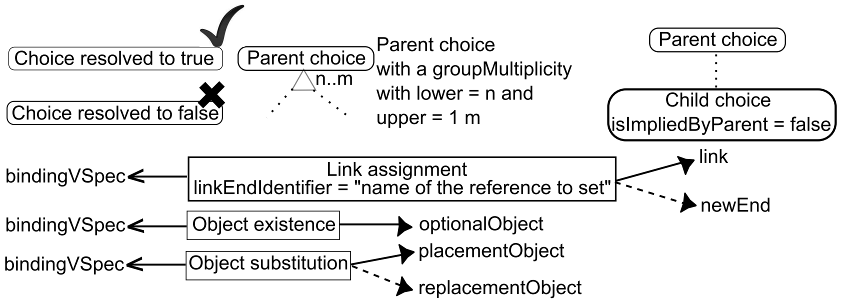


Figure 6: CVL concrete syntax

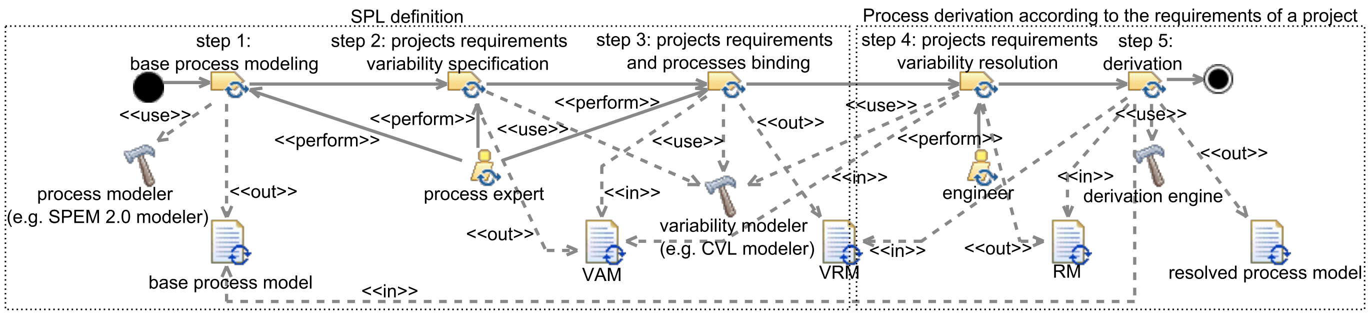

SPL definition
Process derivation according to the requirements of a project
step 1: base process modeling
step 2: projects requirements variability specification
step 3: projects requirements and processes binding
step 4: projects requirements variability resolution
step 5: derivation
<<use>>
<<perform>>
<<perform>>
<<use>>
<<perform>>
<<use>>
<<out>>
<<use>>
<<perform>>
<<use>>
<<out>>
process modeler (e.g. SPEM 2.0 modeler)
<<out>>
process expert
<<in>>
<<in>>
engineer
<<in>>
derivation engine
<<out>>
variability modeler (e.g. CVL modeler)
VAM
VRM
<<in>>
<<out>>
RM
resolved process model
base process model
<<in>>

Figure 7: Overview of an approach using CVL to bind the requirements variability to the process variability

the following of this section, we detail the different steps of our approach, ranging from the SPL definition to the process derivation. We illustrate them thanks to the illustrative example introduced in Section II.

A. The SPL Definition

1) Methodology for Process Elements Modeling (step 1): This step allows the process expert to model the process elements required to define the expected family of processes. The process expert performs this step using any modeler based on its favorite SPML.

The process expert first models the most often used process of the company into a base process model (called base model in CVL). Then, the process expert provides all other process elements that do not belong to the most often used process (called *external process elements*). These external process elements are added in the same base process model, without linking them to the most often used process. Even if several processes use the same external process element, it is modeled only once in the base process model. Consequently, when a process element common to several processes evolves, it has to be updated only once. The different processes that use an external process element often require different settings of the properties of this external process element. In this case, the process expert only sets the properties as the most often used of these processes requires.

Using SPEM 2.0 as SPML, the process expert starts by modeling the method content elements upon which the most often used process and the external process elements will be defined. The process expert then models the most often used process and the external process elements in different delivery processes. This way, there is no operation to perform on the

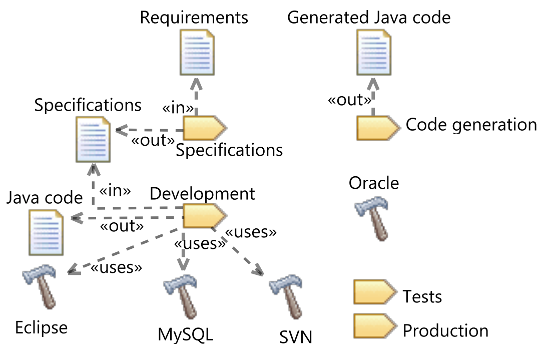


Figure 8: Method content elements of the illustrative example

base process model to derive the most often used process. Indeed, in SPEM 2.0, a process describing a complete project life cycle is described into a delivery process, and there the most often used process is already described in a delivery process.

While Figure 3 shows the most often used process of our Java development process example, Figure 8 shows the corresponding method content elements. As for the external process elements, we model only their properties that correspond to the most often used process that uses these method content elements. For instance, we model that the task definition named *Development* uses the tool definition named *MySQL* and not the *Oracle* one.

Figure 9 shows the process fragment representing the external process elements of the Java development process example. Note that the outgoing control flow from the task use named *Code generation* is invalid w.r.t the metamodel since it does not have any target. Nevertheless, this control flow is useful because getting a Java development process with

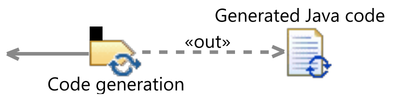


Figure 9: A process fragment representing the external process elements of the illustrative example

code generation consists of adding the task use named *Code generation* to the most often used process, as well as a control flow from the task use named *Code generation* to the task use named *Development*, while redirecting the control flow from the task use named *Technical and Functional Specifications* to the task use named *Code generation*. Since most of the process modelers (*e.g.,* SPEM-Designer[7]) do not enable the modeling of invalid process elements, we rely on approaches that handle model fragments (*e.g.,* [6]) for such a purpose.

2) Projects Requirements Variability Specification (step 2): In the second step of our approach, the process expert uses the CVL variability abstraction metamodel to specify the projects requirements variability in the VAM.

The right part of Figure 10 shows the VAM of the requirements for the different Java development projects of our illustrative example. A project uses a database or not, and if yes, it is either a MySQL one or an Oracle one. Moreover, a project uses code generation or not to produce the code of the application to deliver.

In addition to the concepts used in the illustrative example, CVL enables the specification of variability about the number of fragment instances. Furthermore, CVL enables the expression of constraints on the variability resolution that are not in a parent-child relationship. To this end, CVL provides the way to express constraints using first order logic (including universal and existential quantifications), as well as arithmetic constraints. CVL also enables to automatically infer the resolution of variability specifications from the resolution of other variability specifications, according to the constraints imposed on their resolution.

3) Projects Requirements and Processes Binding (step 3): In the third step of our approach, the process expert defines in the VRM the binding between projects requirements variability (the VAM defined in step 2) and processes (the base model defined in step 1).

The center part of Figure 10 shows the VRM of our illustrative example. This VRM specifies the configuration of a delivery process used by an engineer according to the possible requirements. The activity named *Java development process* represents the delivery process of the most often used process. For a project with a MySQL database and without code generation, the VRM does not modify the delivery process that contains the most often used process (cf. Figure 3). For a project with an Oracle database, it replaces the tool definition named *MySQL* by the one named *Oracle*. For a project without database, it deletes the tool definition named *MySQL*. For a project with code generation, the VRM adds the task use named *Code generation*, its outgoing control flow, and the work product use named *Generated Java code* to the delivery process of the most often used process. It then redirects the outgoing control flow from the task use named *Specifications* to the task use named *Code generation* and redirects the outgoing control flow from the task use named *Code generation* to the task use named *Development*. It finally puts the work product use named *Generated Java code* as input of the task use named *Development*. To this end, the VRM adds a process parameter to the task use named *Development* and puts the work product use named *Generated Java code* as parameter type of this process parameter.

In addition to the concepts used in the illustrative example, CVL provides various constructs to specify the VRM that are useful in the context of processes. A construct assigns the value of an attribute of a process element (slot assignment). Some constructs specify the existence of an attribute or of a link of a process element (slot value existence, link existence). Constructs enable to specify a value to assign or an object to substitute into the RM (parametric link assignment, parametric slot assignment, parametric object substitution). These constructs are useful when the process expert does not know the variants of a process element at the time of the VRM's edition. Some constructs replace a process fragment by another one (fragment substitution, repeatable fragment substitution) and create several instances of a process fragment (repeatable fragment substitution). Figures 11a and 11b respectively illustrate a fragment substitution and a repeatable fragment substitution. On the other hand, some process metamodels enable the reuse of objects. For instance, SPEM 2.0 introduces the notion of process pattern, which factorizes a process fragment common to several activities. Then, these activities can reuse this process pattern. When a process pattern contains variability, different activities can require different resolutions of this process pattern to reuse it. However, an activity cannot reuse a resolved process pattern if it is not resolved appropriately. This problem applies to any object that contains variability and that is reused with different variability resolutions in a model. The CVL specification calls such an object a configurable container object. To this end, CVL introduces constructs that specify a configurable container object (configurable unit) and that clone the configurable container object, resolve variability on the clone and finally redirect the link of the base model object (that uses the resolved clone) to the resolved clone (configurable unit usage). Figure 11c illustrates the configurable unit and configurable unit usage variation points with the example of process patterns. Finally, the process expert can define new variation points using a model to model transformation (opaque variation point).

B. Process Derivation According to the Requirements of a Project

1) Projects Requirements Variability Resolution (step 4): The engineer uses the CVL resolution metamodel to select in an RM the requirements of a given project among those that the VAM specifies. We refer to this action as projects requirements variability resolution.

[7]http://marketplace.eclipse.org/content/spem-designer-helios-version

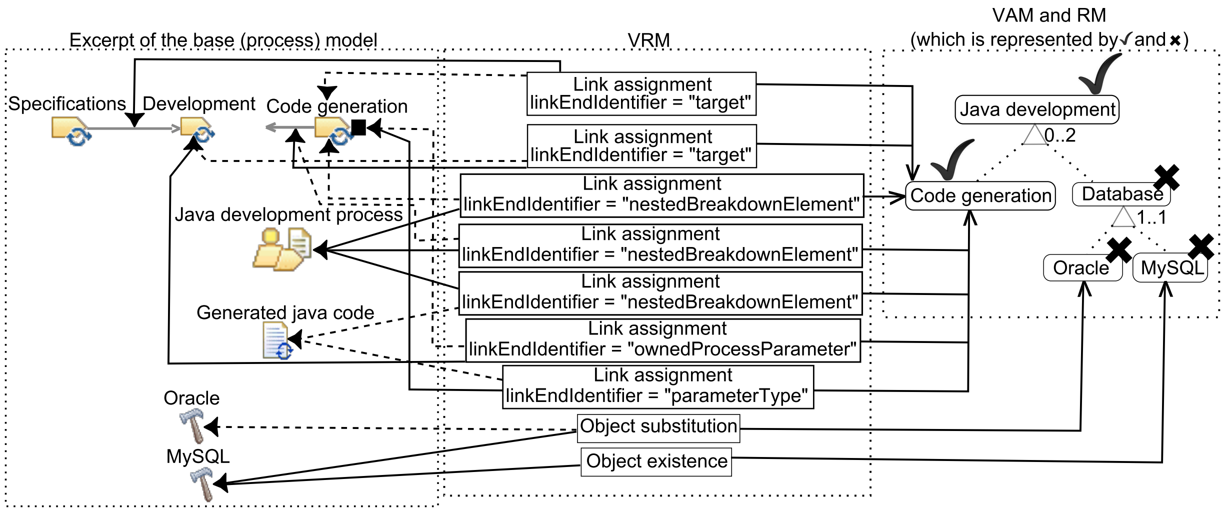
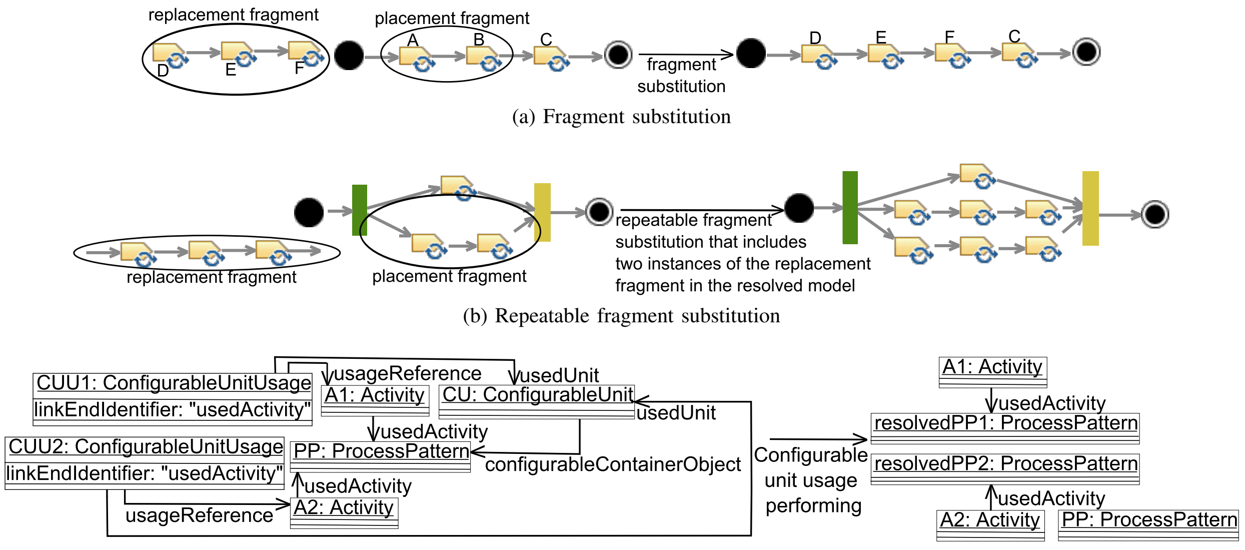

Excerpt of the base (process) model
VRM
VAM and RM
(which is represented by ✓ and ✖)
Specifications
Development
Code generation
Java development process
Generated java code
Oracle
MySQL
Link assignment
linkEndIdentifier = "target"
Link assignment
linkEndIdentifier = "target"
Link assignment
linkEndIdentifier = "nestedBreakdownElement"
Link assignment
linkEndIdentifier = "nestedBreakdownElement"
Link assignment
linkEndIdentifier = "nestedBreakdownElement"
Link assignment
linkEndIdentifier = "ownedProcessParameter"
Link assignment
linkEndIdentifier = "parameterType"
Object substitution
Object existence
Java development
0..2
Code generation
Database
1..1
Oracle
MySQL

Figure 10: The VAM, the VRM, and one possible RM of the illustrative example



(a) Fragment substitution

(b) Repeatable fragment substitution

(c) Configurable unit and configurable unit usage

Figure 11: Overview of other CVL variation points

The right part of Figure 10 also shows an RM for a Java development project that requires code generation and no database (see $\surd$ for the choices selected by the engineer and **X** for the implicitly unselected ones) .

2) Automatic Process Derivation (step 5): Finally, the last step of our approach consists in automatically deriving the process corresponding to the requirements of a given project. For this purpose, we use the CVL derivation engine to allow an engineer to derive a new process model from the base process model (step 1), and according to the requirements selected in the RM (step 4), and the variability realization captured in the VRM (step 3). This step produces a resolved process model.

Figure 12 shows the resolved process model corresponding to the RM depicted in Figure 10. In the resolved process model, the CVL derivation engine has deleted the database and it has introduced the code generation task to the most often used Java development process.

If there are unlikely recurrent project specific needs, the engineer can manually adapt the derived process to these needs. Indeed, there is no need to capitalize on project specific needs into the process line if they are unlikely recurrent.

V. Use of CVL to Bridge the Gap Between the Requirements Variability and the Processes Variability

In this section, we answer rq2 by discussing the capacity of our CVL-based approach to capture the processes variability according to the requirements variability.

According to our approach, the VAM captures the requirements variability, while the VRM specifies the binding between the requirements variability and the base process model. These three models (VAM, VRM and base process model) thus constitute the definition of an SPL.

Then, CVL allows an engineer i) to select the requirements of a given project in a realization model (RM), and ii) to automatically derive a process model from a given RM and the base process model.

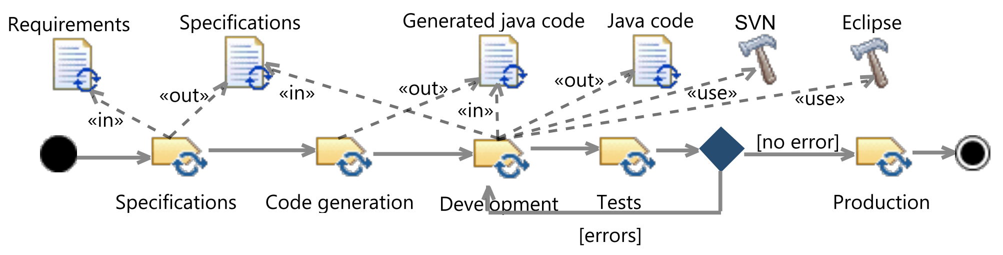


Figure 12: Complete resolved model according to the RM of Figure 10

In our illustrative example, we were able to capture the requirements of 384 Java development projects of the Sodifrance company and their corresponding processes. The Sodifrance's Java development projects vary on the version control system (SVN or CVS), the database (MySQL, Oracle, Postgresql, or no database), the GUI framework (Struts, JSF, Flex, or GWT), the build tool (Ant or Maven), the development (with code generation or not) and the delivery (delivery of the source code, of the compiled code or installation of the application on the customer environment by a Sodifrance engineer). Following our approach, the SPL is composed of 86 model elements (48 SPEM process elements, 21 Variability Specifications and 17 Variation Points). In comparison, the modeling of the different processes in extension (i.e., without factorizing the common parts between processes) would have required the modeling of 384 processes, each of them made of at least 30 process elements. Thus, it would have led to the modeling of at least 11 520 process elements.

Basic Feature Models (BFMs) [7] and Orthogonal Variability Models (OVMs) [1] would also enable the specification of the projects requirements variability thanks to a dedicated variability model. However, they do not provide a mechanism to link the variability model (here, the requirements) to the design models (here, the processes). On the contrary, the VAM enables the capture of the requirements variability and the VRM enables to directly reflect this variability on processes.

VI. Use of CVL Independently of the Process Metamodel

In this section, we answer rq3 by analyzing the independence promoted by CVL with respect to the metamodel on which it is used (in our case, SPEM).

Since CVL uses a string in the object handle and the link handle to reference a model element, CVL is independent of the language on which it is applied. Nevertheless, we observe that the CVL derivation operator can produce an invalid resolved process model, while the VAM, the VRM and the RM are valid. In the following, we identify the possible sources of such an invalidity, and we classify them according to the way to prevent them.

The first errors we observe occur during assignments (i.e., link assignment, slot assignment, parametric link assignment, parametric slot assignment) or substitutions (i.e., object substitution, parametric object substitution, fragment substitution, repeatable fragment substitution) that do not respect the type compatibility with the base model.

These possible errors constitute the first category we have identified. These errors arise because CVL does not constrain the specification of the VRM according to the metamodel to which it applies. Nevertheless, the use of CVL could be forced by constraints generically expressed on its metamodel, thus ensuring the validity of the VRM with respect to the resolved models. For example, the type compatibility mentioned above in the context of a link assignment could be avoided by using the following constraint expressed with the Object Constraint Language (OCL) [8] on the CVL metamodel:

```
context LinkAssignement inv:
  -- where 'find' resolve an URI into the suitable
       model element and 'OclType' gives the type of
       the model element on which it is applied.
  find(self.newEnd.MOFRef).OclType =
    find(self.link.MOFRef).OclType
```

The second error is when there are dandling references in the resolved model, *i.e.* when a link refers to an object that does not exist anymore. This occurs during an object existence, when an object is deleted but not its incoming links. In order to avoid this error, the following constraint must be satisfied: when an object is deleted, its incoming links must be deleted. A link existence enables the deletion of the incoming links. We are going to see in the following paragraph the error related to the link existence and how to avoid it.

The third error concerns the non-respect of the multiplicity of a reference. For instance, when a reference with a lower bound of n and a upper bound of p is instantiated by m links, with $m < n$ or $m > p$. This occurs when a (parametric) link assignment (respectively a link existence) creates (respectively deletes) a link that makes the number of links instantiating a reference greater (respectively lower) than the upper (respectively lower) bound of this reference. This also occurs during a fragment substitution, because CVL enables the assignment of new outgoing and incoming links to the replacement fragment, as well as the deletion of existing links, without ensuring the respect of the multiplicity of the references that the links instantiate. Moreover, the error occurs during a repeatable fragment substitution, when the incoming links to which the replacement fragments are bound cannot reference as much replacement fragments as the RM defines. Finally, this occurs during a configurable unit usage, when the container of the configurable container object cannot contain as much configurable container objects as the ones that are created by configurable unit usages. In order to avoid this error, constraints must be satisfied. A new link can instantiate a reference whose upper bound is strictly upper than one only if the number of links already instantiating this reference is strictly lower than the upper bound of the reference. A link can be deleted only if the reference it is instantiating before its deletion has a number of links strictly upper than its lower bound. A process fragment can be duplicated only if its incoming links can also be duplicated while ensuring the respect of the multiplicity of the reference they instantiate. For a configurable unit usage, the container of the configurable container object must be able to contain one more configurable container object.

The second and third errors constitute the second category we have identified. These errors occur because CVL does not constrain the specification of the VRM according to the base model to which it applies and according to the metamodel of the base model. A solution to ensure the constraints of the second and third errors would be a generic static analysis tool that would check before the derivation if the constraints are satisfied. By generic we mean metamodel independent. If the constraints are not satisfied, the tool would inform the engineer of the error, of the variation point that introduces it and of the constraint that is violated. Then, the process expert and the engineer would have to correct the errors to start the derivation.

The fourth error is when the resolved process model conforms to its metamodel but is semantically inconsistent. Figure 13 illustrates this error. In SPEM 2.0, a work sequence specifies the dependencies about the execution of activities. The work sequence of kind *finishToFinish* means that the activity named *B* can finish when the activity named *A* finishes. The work sequence of kind *finishToStart* means that the activity named *B* can start when the activity named *A* finishes. Here the work sequence of kind *finishToFinish* is useless. In order to avoid this error, we must ensure that there is no semantic inconsistencies in the resolved process model.

This fourth kind of error constitutes the third category we have identified. As for the second category, these errors also occur because CVL does not constrain the specification of the VRM according to the base model to which it applies and according to the metamodel of the base model. However, in this case, the errors are specific to a metamodel and cannot be generalized. A solution would be to implement a SPEM-specific derivation engine. It would ensure during derivation that SPEM-specific constraints are satisfied, on top of performing the same operations than the CVL derivation engine. When a SPEM-specific constraint is not satisfied, the SPEM-specific derivation engine would inform the engineer of the error, of the derivation step that has introduced the error and of the constraint that is violated.

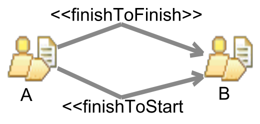


Figure 13: Over-specified work sequence

VII. Related Work

One approach for reusing processes provides a technique for the retrieval of processes stored into a repository [9]. The Debus-Booch prototype [10] enables to select a software design process from a family of such processes and to execute it. These approaches define the different processes in extension. Therefore, when a part common to several processes evolves, it has to be updated in all the processes it belongs to, which is error prone and time consuming.

The following approaches address this problem by defining an SPL, in order to model places in processes that vary and the possible variations. However, they are dependent of the process metamodel because they modify the process metamodel with variability mechanisms. In the field of software processes, one approach relies on the variability mechanisms that SPEM 2.0 provides in order to model a general process model with variability [11]. The vSPEM approach [12] extends SPEM 2.0 with variability mechanisms empirically evaluated as more understandable [13]. Still in the field of software processes, another approach [3] provides a metamodel for an SPL, that must be specialized according to the process metamodel and the variability to capture. In the field of business processes, approaches extend the Event-driven Process Chain (EPC) process metamodel to model variability into an EPC reference process. The reference process captures all the different processes using conditional branching. Configurable EPC (C-EPC) [14] expresses variability on the functions and connectors EPC model elements. The approach in [15] extends C-EPC with the concepts of role and object and specifies variability on these concepts. Aggregate EPC (aEPC) [16] links process elements to the configurations that use them. The PROcess Variants by OPtions (Provop) approach [17] copes with the non distinction between branching nodes that are part of a process and the ones that denote different processes in a reference process. It proposes to model one process and the operations to perform on it to derive the different processes. Still in the field of business processes, approaches [18], [19] extend the BPMN metamodel for modeling places where variability occurs in processes and their possible resolutions. Another approach [20] captures process variability into a hierarchical structure and provides support for variant process elements management and reuse. Other approaches [21]–[23] partially address the problem of the dependence towards the process metamodel by transforming a process model into a pivot structure to define variability. However, these approaches require the definition of a transformation for each process metamodel. Finally, the Adaptive Business Process Modeling in the Internet of Services (ABIS) approach [24] introduces constructs for managing variability in BPMN 2.0 processes without modifying the BPMN 2.0 [25] metamodel. This approach is also dependent of the process metamodel. Indeed, the variability constructs need to be adapted in case of the use of another process metamodel.

The following approaches specify variability independently of the process metamodel. However, they do not provide all the mechanisms for automatically deriving a process. One approach [26] proposes to model process variability separately from the process model, using BFMs or OVMs. However, BFMs do not provide a binding mechanism between the BFM and the process model. OVMs provide a binding mechanism with the model for which variability is specified (*e.g.,* the process model). However, this binding mechanism implies modifying the metamodel of the model for which variability is specified. Furthermore, OVMs do not provide a mechanism to automatically derive a process. Other approaches are similar to ours. One approach [27] proposes to model one process and to augment it with process elements selected according to a process goals specification, in order to obtain a particular process. The notion of process goals specification is close to our notion of requirements of a project. Our approach goes further by providing mechanisms to automate the derivation of a process according to the requirements of a project. Another approach [28] provides a questionnaire-driven method for selecting a process from a reference process model. Actions are performed on the reference process model to derive a process according to the answers to the questions. Our approach goes further by providing a language to define actions on process models whatever their metamodel is.

To conclude, approaches for reusing processes to our knowledge either define the different processes in extension, or are dependent of the process metamodel, or do not provide all the mechanisms for automatically deriving a process.

VIII. Conclusion and Perspectives

We propose an approach to use CVL in the context of processes and we perform an experiment (i) to understand how to use this new variability modeling language in this context, (ii) to discuss if CVL enables the management of processes variability, (iii) and to discuss if CVL enables the management of processes variability while being independent of the process metamodel. The lessons learned from this experiment are mainly that CVL enables the definition of an SPL and the automatic derivation of a process from this SPL according to the requirements of a project. CVL is also independent of the process metamodel, but this can be a source of errors in the resolved process model.

Using CVL to model process lines has the advantage of being independent of the process metamodel. This enables the reuse of the approach with every process metamodel as well as the reuse of process metamodel specific tools. The current limitation of the use of CVL for modeling process lines is that models are difficult to edit and maintain. Indeed, processes other than the most often used one are not visible in the SPL due to the structure of the base process model and the variability management. Implementing a tool that enables processes and variability modeling into the same diagram would address this limitation.

As perspectives of work, we are implementing a generic static analysis tool and a derivation engine plugin for SPEM, in order to automatically ensure the satisfaction of the constraints for deriving a valid resolved process model.

References

[1] K. Pohl, G. Böckle, and F. J. v. d. Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer, 2005.

[2] H. D. Rombach, "Integrated software process and product lines," in *ISPW*, 2005, pp. 83–90.

[3] T. Ternité, "Process Lines: A Product Line Approach Designed for Process Model Development," in *SEAA*, 2009, pp. 173–180.

[4] OMG, "Software and Systems Process Engineering Metamodel Specification (SPEM) Version 2.0," http://www.omg.org/spec/SPEM/2.0/, 2008.

[5] ——, "Documents Associated With UML Version 2.0," http://www.omg.org/spec/UML/2.0/, 2005.

[6] R. Ramos, O. Barais, and J. Jézéquel, "Matching model-snippets," in *MoDELS 07*, 2007, pp. 121–135.

[7] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study," Carnegie-Mellon University Soft. Eng. Institute, Tech. Rep., 1990.

[8] OMG, "Documents associated with Object Constraint Language, Version 2.2," http://www.omg.org/spec/OCL/2.2/, 2010.

[9] R. Lu and S. Sadiq, "On the Discovery of Preferred Work Practice Through Business Process Variants," in *ER*, 2007, pp. 165–180.

[10] X. Song and L. J. Osterweil, "Engineering Software Design Processes to Guide Process Execution," *IEEE Transactions on Software Engineering*, vol. 24, no. 9, pp. 759–775, 1998.

[11] J. Hurtado Alegría, M. Bastarrica, A. Quispe, and S. Ochoa, "An MDE Approach to Software Process Tailoring," in *ICSSP*, 2011, pp. 43–52.

[12] T. Martínez-Ruiz, F. García, and M. Piattini, "Towards a SPEM v2.0 Extension to Define Process Lines Variability Mechanisms," in *SERA*, 2008, pp. 115–130.

[13] T. Martínez-Ruiz, F. García, M. Piattini, and J. Münch, "Modelling Software Process Variability: an Empirical Study," *IET Software*, vol. 5, no. 2, pp. 172–187, 2011.

[14] M. Rosemann and W. M. P. van der Aalst, "A Configurable Reference Modelling Language," *Information Systems*, vol. 32, no. 1, pp. 1–23, 2007.

[15] M. Rosa, M. Dumas, A. H. Hofstede, J. Mendling, and F. Gottschalk, "Beyond Control-Flow: Extending Business Process Configuration to Roles and Objects," in *ER*, 2008, pp. 199–215.

[16] H. A. Reijers, R. S. Mans, and R. A. van der Toorn, "Improved Model Management with Aggregated Business Process Models," *Data Knowledge Engineering*, vol. 168, no. 2, pp. 221–243, 2009.

[17] A. Hallerbach, T. Bauer, and M. Reichert, "Capturing Variability in Business Process Models: the Provop Approach," *Software Maintenance*, vol. 22, no. 67, pp. 519–546, 2010.

[18] V. Kulkarni and S. Barat, "Business Process Families Using Model-Driven Techniques," in *BPM*, 2010, pp. 314–325.

[19] A. Schnieders and F. Puhlmann, "Variability Mechanisms in E-Business Process Families," in *BIS*, 2006, pp. 583–601.

[20] W. Derguech and S. Bhiri, "Reuse-Oriented Business Process Modelling Based on a Hierarchical Structure," in *BPM*, 2010, pp. 301–313.

[21] F. Gottschalk, W. M. van der Aalst, M. H. Jansen-Vullers, and M. La Rosa, "Configurable Workflow Models," *Cooperative Information Systems*, vol. 17, no. 2, pp. 177–221, 2008.

[22] S. Meerkamm, "Configuration of Multi-perspectives Variants," in *BPM*, 2010, pp. 277–288.

[23] E. Santos, J. Castro, and O. Sánchez, J.and Pastor, "A Goal-Oriented Approach for Variability in BPMN," in *WER*, 2010, pp. 17–28.

[24] M. Weidmann, F. Koetter, M. Kintz, D. Schleicher, and R. Mietzner, "Adaptive Business Process Modeling in the Internet of Services (ABIS)," in *ICIW*, 2011, pp. 29–34.

[25] OMG, "Documents Associated with Business Process Model and Notation (BPMN) Version 2.0," http://www.bpmn.org/, 2011.

[26] J. Simmonds and M. C. Bastarrica, "Modeling Variability in Software Process Lines," Universidad de Chile, Tech. Rep., 2011.

[27] B. I. Simidchieva, L. A. Clarke, and L. J. Osterweil, "Representing Process Variation with a Process Family," in *ICSP*, 2007, pp. 109–120.

[28] M. La Rosa, J. Lux, S. Seidel, M. Dumas, and A. ter Hofstede, "Questionnaire-driven Configuration of Reference Process Models," in *CAiSE*, 2007, pp. 424–438.