Model-driven support for business process families with the Common Variability Language (CVL)

Daniel Calegari, Andrea Delgado, Leonel Peña

Universidad de la República, Facultad de Ingeniería,
Montevideo, Uruguay, 11300,
{*dcalegar, adelgado, lpena*}@fing.edu.uy

Abstract

To achieve a business objective, organizations may require variants of the same business process that depend on the context in which they are enacted. Several proposals have emerged to deal with the variability of business processes, focused on the modeling of a so-called process family. The proposals try to avoid modeling each variant separately, which implies duplication and maintenance of the common parts. Few of them also focus on automatically deriving a process variant from the definition of a process family, which is a central and complex task. One of these proposals is the Common Variability Language (CVL), which allows representing variability transparently in a host language. In this extended article we present a proposal integrating CVL together with the Business Process Model and Notation (BPMN 2.0) for modeling business process families, and the use of Model-Driven Engineering (MDE) techniques for the automated generation of specific process variants. We also describe the integrated tool support we provide for the complete approach, the proposal validation with a case study from a project we carried out within our university, and an evaluation by using the VIVACE framework.

Keywords: Business process families, variability, Common Variability Language (CVL), Business Process Model and Notation (BPMN), Model Driven Engineering (MDE)

1 Introduction

To support the definition, control, and continuous improvement of business operation, many organizations adopt a business process vision, i.e., the identification of the set of activities that are performed in coordination within an organizational and technical environment to achieve a business objective [1]. Business Process Management (BPM) [2, 1, 3] offers a framework to support the business process lifecycle from modeling, through developing, deploying, executing, and to the evaluation of their execution. Process-Aware Information Systems (PAIS) [4] were traditionally focused on the support of predictable and repetitive business processes, flexibility is required in many scenarios [5], e.g., when tackling with processes that cannot be fully prescribed (looseness), that must be adapted at run time (adaptation), or that will evolve (evolution), or can be adapted at design-time (*variability*). The latter scenario, in which this work focuses on, deals with design-time adaptation depending on specific business requirements of the context in which they will be enacted, e.g., different selling processes depending on the kind of products or payment method, or different accountability steps depending on the country.

This leads to the definition of a so-called business process family [6], in which many business processes are considered variants of a common business process. To define a process family, it is required: to define a common base process (also known as customizable or configurable), to identify what parts of such process may vary according to a specific context (known as variation points), to specify the alternatives that the process family may accept (known as variants), and to state the conditions over which each alternative must be selected (known as a configuration). By defining a configuration, which leads to the selection of a specific variant for each one of the variation points, a concrete process model within the family (i.e., a process variant) can be derived. Although process modeling languages usually do not support the modeling of process families, e.g., the Business Process Model and Notation (BPMN 2.0 [7]) standard, there are many proposals [6, 8] extending such languages, or even providing a language-independent way of representing a process family [9]. Two major variability approaches are usually followed: by restriction or by extension.

In the variability by restriction approach (also known as behavioral approach) all the possible behaviors are represented within the base process, and each process variant is derived by restricting such behavior. An example of this approach is BPMNext [10], a proposal of our own based on an adaptation of BPMN 2.0 for supporting multi-level variation points for activities and roles. In contrast, in the variability by extension approach (structural approach) a set of artifacts separately represent different aspects of the process family, and each process variant is derived by extending the behavior of the base process. An example of this approach is the Common Variability Language (CVL) [11], and its successor the Base Variability Resolution approach (BVR) [12], which are very powerful languages that allow specifying and resolving variability on any other Domain-Specific Language (DSL), e.g., BPMN 2.0, without modifying the host language.

Business process family modeling is essentially a manual task. However, the derivation process (also known as customization) can be achieved automatically. To improve automation by reducing technical complexity to final users, the Model-Driven Engineering (MDE) [13] paradigm can be considered as a promising approach. Within the software development life-cycle envisioned by MDE, there are two basic building blocks: models representing different views of the system to be constructed, and model transformations providing an automatic construction process. Modeling languages are defined using a metamodeling language like the standard Meta-Object Facility (MOF, [14]), as do BPMN 2.0 and CVL/BVR, whilst model transformations can be expressed using high-level languages like the ATL [15] language and toolkit.

There are some proposals, e.g., BPMNext, which provides a model-driven strategy for the customization process. However, in most of the variability proposals, this process is carried out following a not well defined, manual, error-prone, and complex process, since languages are not formally defined, the customization process is expressed using examples, or supporting tools are not available. This happens for all variability by extension approaches, e.g., in the case of CVL/BVR, the language is formally defined, but there is neither a concrete procedure nor tool support for achieving the customization process.

In this article, we complement the work started with BPMNext, with a model-driven strategy for the customization process of business process families following a variability by extension approach, i.e., the CVL/BVR approach. Our contribution is threefold. First, we analyze how CVL/BVR can be used with BPMN 2.0 to represent a process family and automatically derive process variants by the execution of a model transformation. Second, we integrate these ideas into an Eclipse-based editor and validate our approach through an application taken from the variability literature and a case study from a real project we carried out within our university [16]. Besides, we provide an evaluation of the approach using the VIVACE framework [17] for assessing and comparing process variability approaches, and the support they provide for process variability throughout the business process life-cycle. The tool we have developed and the case study can be found in [18].

This paper is a substantially extended and thoroughly revised version of [19]. We extend the aforementioned work by providing:

- a case study (Section 4.2) related to the "Assignment of positions in academic exchange programs" process family, from a real project we carried out [16] within our university, and a discussion regarding the results of the execution of the approach.
- an extended discussion (Section 6) about the use of other CVL variation mechanisms for improving the proposal.

The rest of this paper is organized as follows. In Section 2 we summarize and discussed related and previous work. In Section 3 we describe the model-driven strategy which uses CVL/BVR together with BPMN 2.0 for modeling business process families and deriving process variants, and we introduce the Eclipse-based editor we have built. Then, in Section 4, we describe the application of the proposal in two different contexts, and in Section 5 we report the evaluation of the proposal using the VIVACE framework. In Section 6 we study how other CVL/BVR variation points can be used to improve the proposal. Finally, in Section 7 we present some conclusions and an outline of future work.

2 Related Work

In [6, 8] the authors performed comprehensive surveys about approaches for modeling process families. In [10] we delve into approaches using BPMN 2.0 as a standard specification language, as well as language-independent approaches that were adapted to be used with BPMN 2.0. In Table 1 there is a summary of the approaches, in particular: how the variability is expressed (Restriction/Res or Extension/Ext), the models required for the specification, the kind of elements over which the variability is expressed, the base language used (BPMN 2.0 extension or language-independent), and their corresponding tool support (tool availability up to this date).

Table 1: Comparison of different approaches (extended from [10])

Approach	Var kind	Representation	Variation points	Language	Tool
BPMN* [20]	Res	Base model + Feature model	Any process element	BPMN 2.0 ext	✗
BPMNext [10]	Res	Integrated model	Activities, roles	BPMN 2.0 ext	✓
BPMNt [21]	Ext	Tree-based representation	Any process element	BPMN 2.0 ext.	✗
C-BPMN [22]	Res	Integrated model	Activities, gateways	BPMN 2.0 ext.	✗
CVL [11] / BVR [12]	Ext	Base model + Context model	Any process element	Language ind.	✓
PESOA [23, 24]	Res	Base model + Feature model	Activities, data	Language ind.	✓
Provop [25]	Both	Base model + Context model	Activities	Language ind.	✗
vBPMN [26]	Ext	Base model + Context model	Activities, gateways, events	BPMN 2.0 ext.	✗

BPMN* [20] is BPMN 2.0 extension providing a variability by restriction approach. It consists of adding stereotypes and tags to every process elements (activities, data, roles, gateways, events), which are related to a feature model. It is a simple approach with great expressiveness. As in the case of the other variability by restriction approaches, it has a scalability problem related to the specification of alternatives in the same model. Tool support is not available.

BPMNext [10] is also a BPMN 2.0 extension providing a variability by restriction approach, that is part of our work on process variability for BPMN 2.0. The approach is based on the ideas provided by the vSPeM approach for software processes [27]. In BPMNext, variation points are defined within the process, supporting multi-level variation points for activities and roles. In contrast with the last approach, variants are not expressed in the same model, but in complimentary BPMN 2.0 models. When deriving a process variant, using a model transformation, each variation point is substituted with exactly one variant. Tool support is available.

BPMNt [21] is a conservative extension of BPMN 2.0. i.e., not affecting its base semantics, providing a variability by extension approach. Despite its great potential, it has a poor graphical representation which affects comprehension (uses a tree-based representation of a process model). For each variation point, it is possible to express deletion, replacement, and adding of variants, as well as relations between them. Tool support is not available.

Configurable-BPMN (C-BPMN [22]) provides a variability by restriction approach focused on BPMN 2.0. It defines configurable fragments, which are structured blocks that are equipped with expressions that define context conditions. Process customization is achieved by using a questionnaire that expresses a specific context configuration. It is a very expressive approach since it allows variation point on activities, data, roles, and gateways, but as a process modeling language, is less expressive than BPMN 2.0. In this case, the variation point can be expressed on activities and gateways. Tool support is not available.

Process Family Engineering in Service-Oriented Applications (PESOA [28]) project provides a language-independent variability by restriction approach which defines the notion of a variant-rich process model, i.e., a process model that contains every possible variant of the process family. The overload of annotations together with the need of expressing every variant in the same process affects the comprehension of the process family. The approach also uses a feature model that determines the context conditions to which each variant applies. The approach is supported by an available tool, only providing modeling capabilities, not supporting the customization process. The approach was adapted for BPMN 2.0 in [23, 24].

PROcess Variants by OPTIONS (Provop, [29]) is a language-independent variability by restriction/extension approach. The language allows the definition of configurable regions and a set of configuration alternatives together with context rules that restrict the application of such alternatives. This approach provides a very simple, modular, and expressive mechanism for variability modeling. However, the language only admits variation points concerning activities (tasks and subprocesses). Moreover, the modeling tool is not currently available. The approach was adapted for BPMN 2.0 in [30, 31, 32].

vBPMN [26] is another variability by extension approach. The approach combines BPMN 2.0 models with the rule language R2ML for managing variabilities on activities, gateways, and events at runtime (what is called flexibility-by-design [5]). However, R2ML adds complexity to the definition of the process family. The configurable process is built together with a data context and adaptive workflow segments (structured block of the model) which depends on the context. Tool support is not available.

Finally, the Common Variability Language (CVL, [11]) is a language-independent variability by extension approach. The extension mechanism separates the specification of the base model and its variants, achieving better readability, greater comprehension, and scalability. The approach also uses a variation model defining which variants are used in each variation point, and a resolution model that specifies contextual conditions that must apply for configuring a model. The Base Variability Resolution models approach (BVR) [12] was built on the revised submission of CVL but is simplified and enhanced relative to that language. This

approach is supported by available tools just for modeling issues. The approach was adapted for BPMN 2.0 in [9]. The authors briefly depict how CVL and BPMN can be used together for expressing variation points on activities, besides the language can potentially involve other elements. Their work is focused on how to process variants that could be automatically adapted at run-time. Its prototype system is not available.

Complimentary to our work defining a variability by restriction approach, i.e., BPMNext, we focused on variability by extension approaches. These approaches, i.e., BPMNt, Provop, vBPMN, CVL/BVR, only provide modeling capabilities, not supporting the customization process, and they lack tool support. In this context, we take CVL/BVR, a very powerful modeling language providing a basic tool, and we go much deeper than previous works [9], defining, implementing, and evaluating a model-driven strategy focused on describing the connection between both languages (CVL/BVR and BPMN 2.0) and providing the automatic generation of variants at design time. In the following sections, we describe such results.

3 CVL/BVR variability proposal for BPMN

In what follows we present an MDE strategy for the modeling of process families and the derivation of process variants through model transformations. We follow a variability by extension approach by applying the CVL/BVR proposal to BPMN 2.0-based process families. We first present the CVL/BVR approach and then we define how it is applied for BPMN 2.0. This application also involves the definition of a model transformation taking a CVL/BVR + BPMN 2.0 specification and producing a BPMN 2.0 compliant process variant, and the construction of a tool that allows the modeling of BVR+BPMN 2.0 specifications, as well as the execution of the transformation. Concrete examples to fully understand the proposal are described in detail in Section 4.

3.1 Introducing CVL & BVR

The Common Variability Language (CVL) [11] is a language that is now a Revised Submission in the Object Management Group. The language has been defined as a domain-independent language for specifying and resolving variability. The Base Variability Resolution models approach (BVR) [12] is built on CVL, simplified, and improved due to industrial needs. It also defines more specific languages for expressing some parts of the CVL specification and provides tool support. Both follow the schema depicted in Figure 1.

CVL/BVR takes any Domain-Specific Language (DSL) and adds up constructions to manage variability concepts. The schema is composed of four kinds of models: whilst the Variability Model and the Resolution Models are defined in CVL/BVR, the Base Model and Resolved Models can be defined in any language based on the standard Meta-Object Facility (MOF), e.g., the BPMN 2.0 language.

Figure 1: CVL schema from [11]

- A **Base Model** represents a family of models (also called a product line model) from which specific model variants can be described. The base model is described using any DSL, e.g., BPMN 2.0, and contains the definition of the common elements of the family (e.g., a base process), the variants accepted by the family of models (e.g., process fragments), and the variation points where the variants will be substituted according to the context (e.g., process fragments within the base process).
- A **Variability Model**, which is defined in the CVL/BVR specification, contains the definition of the correspondence between the elements of the base model with the variation points (e.g., the relation of a process fragment within the base model and a process fragment that can substitute it), the restrictions, and the context in which the variation points will be valid. This model does not interfere with the base model specification, and its genericity allows it to be applied to any type of language. BVR defines a way of expressing abstract variation points, and how they can be resolved, without depending on concrete elements of the base model. It is based on the definition of a **VSpec** model, a tree-like structure (feature model) representing abstract variation points and logical decisions related to them (called VSpecs) that need to be resolved for each concrete configuration of the family. BVR also defines a realization model that contains the mapping between the abstract variation points within the VSpec and the elements of the base model.
- A **Resolution Model**, also defined in the CVL/BVR specification, provides a particular configuration of the context defined in the variability model to produce a concrete variant, e.g., which fragment of a process must be substituted for another. In BVR, the VSpec model has a corresponding resolution model which provides the information for evaluating the VSpec model and determines whether each kind of variant will materialize or not. This allows a member of the family to be automatically derived.
- A **Resolved Model**, e.g., a concrete process variant can be derived when executing a CVL/BVR specification, i.e., taking the base model and adapting it by considering the variability model concerning a concrete configuration provided by the resolution model.

3.2 Customizing the CVL/BVR proposal

We need to define how to instantiate the four kinds of CVL/BVR models in the context of BPMN 2.0.

Base Model. There must be a base BPMN 2.0 business process with the common flow between all members of the process family. For the modeling of variation points and variants, we use the BVR concept of a fragment, which is a set of elements that may have a single output and a single input element to such a fragment. In our case, a fragment is any connected set of BPMN 2.0 process elements (atomic or with a complex behavior) with only one input and one output flow. Placement fragments (variation points) are defined within the base model, and replacement fragments (variants) are modeled as a BPMN Collaboration Diagram (named the BPMN replacement model). This diagram contains independent processes (within their corresponding Pools), each one representing a specific replacement fragment. This allows us to keep every fragment in the same model and it is easy to differentiate them (each one is in a different Pool). Since each replacement fragment is represented as a process, it implies that they have a start and an end event. However, these events only exist for modeling purposes and to define only one input and one output.

Variability Model. We use a realization primitive called **Fragment Substitution** which involves the definition of a placement fragment in the base model and one or more replacement fragments that can replace them. Fragment substitutions are defined within a BVR realization model, which determines the correspondence between placements of the base model and replacements of the replacement model. Finally, we use a VSpec model for expressing the variability abstractions. In particular, we use a concrete type of VSpec, called **Choice**, that represents a yes/no decision. The combination of choices and the fragment substitution allows replacing a placement fragment (a variation point) with a replacement fragment (variant) when the choice is resolved.

Resolution Model. We use the VSpec resolution model which provides a way of materializing a variability realization, i.e., a fragment substitution depending on a choice. In the case of a Choice within the VSpec model, when it is resolved, it enables a branch of a decision tree or an abstract variation point that must be configured in the resulting family member.

Resolved Model. It is represented as a BPMN 2.0 process model. The existent CVL/BVR tools do not allow to automatically extract a resolved model but provide the elements and the conceptual basis to do so. Based on these elements, we have developed a model transformation for the automatic generation of variants, which is presented in the following section.

In a few words, the process of defining and configuring a BPMN 2.0-based process family consists of the following steps:

1. Definition of the Base Model, which involves the specification of the BPMN 2.0 base process and the definition of the placement fragments (variation points).
2. Definition of the Variability Model, which involves the definition of the BPMN 2.0 replacement model composed by potentially many replacement fragments (variants), the VSpec model (decision tree of abstract variation points), and the BVR realization model (fragment substitutions between the placement and replacement fragments in the context of a VSpec).
3. Definition of the Resolution Model, which is a concrete VSpec configuration.
4. Execution of the model transformation, which takes the base model, performs the substitutions according to the resolution model, and provides a BPMN 2.0 process model (a concrete variant).

3.3 Generation of variants

Unlike [9], we focused on the automatic generation of variants at design time by using a model transformation, instead of a code-based generation. For such purpose, we define a declarative model transformation based on the ATL [15] language and toolkit.

The model transformation takes as input the different BVR+BPMN 2.0 models: a BPMN 2.0 base model, a BPMN replacement model, and a BVR model composed by a VSpec model supplemented with constraints, a realization model (fragment substitutions), and a corresponding resolution model defining the variant selections.

By querying the BVR model using a helper function, it is possible to determine whether a Fragment Substitution needs to be applied, by iterating over the set of VSpecs (only those VSpecs resolved true).

Listing 1: Querying VSpec resolution

```

helper def: getResolvedFragmentSubstitutions:
  Sequence(bvr!FragmentSubstitution) =
    bvr!FragmentSubstitution.allInstances() ->
      select(fs | thisModule.getVSpecsResolvedTrue
        .includes(fs.bindingVSpec));

```

Each transformation rule resolves if the element that matches the rule must be copied or not to the resolved variant. An element will be copied if it belongs to the base model and it is not included in any placement fragment (is part of the common behavior), if it belongs to the base model and belongs to a Fragment Substitution resolved as false (there is no variant to apply), or if it belongs to a replacement fragment that was resolved as true (it must be applied). For this, another helper function is used.

Listing 2: Querying elements to copy in the resolved model

```

helper def: copyElement(element: bpmn!FlowElement):
  Boolean = thisModule.baseFlowElementsToCopy
    .includes(element)
  or thisModule.replacementFlowElements
    .includes(element);

```

The transformation uses abstract rules for copying model elements, following the structure of the BPMN 2.0 metamodel. As an example, an element of type **Event** is a **FlowNode**, so it has incoming and outgoing references. In the same way, an element of type **StartEvent** is a **CatchEvent** that in turn is an **Event**. The mapping of that structure to rules is as follows. Notice that the first rule makes use of the `copyElement(...)` helper function to determine if the element must be copied in the resulting process model. There are other special cases, e.g., for considering when a fragment must be deleted. In such a case we established a modeling convention for expressing deletion, i.e., the replacement fragment will be composed by just an event without any information on its kind. We also need to remove these intermediate elements from the output model, so a second transformation was implemented for cleaning the resolved process variant.

Listing 3: Copying elements to the resolved model

```

abstract rule copyEvents {
  from
    event: bpmn!Event ( thisModule.copyElement(event) )
  to
    eventOut: bpmn!Event (
      id <- event.id,
      name <- event.name,
      incoming <- event.incoming,
      outgoing <- event.outgoing
    )
}
abstract rule copyCatchEvents extends copyEvents {
  from
    event: bpmn!CatchEvent
  to
    eventOut: bpmn!CatchEvent ( ... )
}
rule copyStartEvent extends copyCatchEvents {
  from
    event: bpmn!StartEvent
  to
    eventOut: bpmn!StartEvent ( ... )
}

```

3.4 Tool Support

The tool, that can be found in [18], is based on an integration of the BVR Tool¹ and the Eclipse BPMN2 Modeler². Eclipse BPMN2 Modeler is an open-source graphical business process modeling tool integrated into the Eclipse environment. BVR Tool is also an open-source graphical tool integrated into Eclipse, which provides a way of integrating BVR with any model editor through the implementation of interfaces. The proposed solution is based on the architecture depicted in Figure 2 which comprises two plugins: **BPMN2ModelerAdapter** integrates the BPMN2 Modeler with the BVR Tool, allowing to model Placement and Replacement fragments that will later be part of a Fragment Substitution; and **BPMN Variability** allowing to select the different models within the BVR+BPMN 2.0 specification and executing the ATL transformation.

Figure 2: Architecture of the BVR+BPMN 2.0 Editor

Once a full BVR+BPMN 2.0 specification is defined, the generation plugin is invoked from the drop-down menu bar option **BPMN Variability**. After that, a box is displayed for selecting the models (files) corresponding to the concrete variant to generate. After selecting all the files, the transformation is executed, and the result persisted in a BPMN 2.0 compliant output file. The step by step use of the tool support will be exemplified in Section 4 for the two BP families we worked with.

¹SINTEF BVR Tool: <https://github.com/SINTEF-9012/bvr>

²Eclipse BPMN2 Modeler: <https://www.eclipse.org/bpmn2-modeler/>

4 Proposal validation

For a practical assessment of the approach, we use two different execution cases (can be downloaded from our repository in [18]): the “Airport check-in” process family was taken from [17] and used as a standard case study when applying the VIVACE framework, and the “Assignment of positions in academic exchange program” process family from a real project we carried out within our university, introduced in [16] and used in [10] for our BPMN 2.0 variability extension, BPMNExt.

We differentiate between the approach taken for the two evaluations we carried out. On the one hand, the “Airport check-in” can be seen as a benchmark evaluation, based on a real case that we selected since it is very well known in the variability community and highly used and cited to compare variability approaches. Since it was used in the VIVACE framework to model and evaluate three other proposals, adding the same process modeled with our approach helps us compare our proposal with the ones presented there. It is also presented in the CLEI paper that we are extending, so we kept it as is, for a basic evaluation of the approach.

On the other hand, the “Assignment of positions in academic exchange program” is a case study that we add as part of the extension of the CLEI original paper, and it is based on a process that we worked within a real project from our university, where a real organization was involved and participated in the identification and modeling phases of the BPM effort, and validation of the BP family. For this case, we present the step by step modeling and automated generation of the selected variant, following the steps for applying the CVL/BVR proposal, as presented in Section 3.2 with the tool support we have developed. Finally, a general discussion of the approach for execution of the approach is presented in section 4.3.

4.1 Check-in process family

The process family considers the process every passenger has to go through when checking in at an airport. Numerous variations exist depending on distinguishing factors: the type of check-in (e.g., online, at the counter, etc.), which, in turn, determines the type of boarding card (e.g., electronic versus paper-based); the flight destination (e.g. United States); the type of passenger (e.g., unaccompanied minors and handicapped people might require extra assistance); the type of luggage (e.g., bulk or overweight luggage); and temporal variations regarding the check-in procedure (e.g., possibility to check-in several days or few hours before departure).

As an example, there is a variant describing when the check-in is done online by the passenger. First, the passenger is identified, and a seat is assigned. But since the passenger is flying from Europe to the United States, it requires information about accommodation as well as filling in the electronic system for travel authorization. Finally, an electronic boarding card is printed, and the passenger drops off the luggage at the business class counter. In contrast, another variant represents the check-in process accomplished at the respective counter at the airport, when there is an unaccompanied minor. In this variant, a special seat is assigned, and an extra form is filled in. Also, a copy of the boarding card is required for the relative accompanying the minor to the boarding gate.

We start by creating the base BPMN 2.0 model, depicted in Figure 3, which defines a common process for the variants. Some activities, e.g., **IdentifyPassenger** and **PrintBoarding** are common for every processvariant, whilst others represent variation points, e.g., **VariantAssignSeat**, **VariantExtraInfo**, **VariantPayment** and **VariantLuggage**.

Figure 3: Base process for the “Check-in” family

Because of the modeling limitations of our solution, two artifacts that share different fragment substitution cannot be contiguous in the base model, which is why there is an event between **VariantAssignSeat** and **VariantExtraInfo**. When the transformation is executed and the variant is generated, this artifact is eliminated. The same happens between **VariantPayment** and **VariantLuggage**.

As depicted at the bottom of Figure 4, BPMN 2.0 replacement processes are modeled as independent processes. We can notice the **DLCR** fragment (bottom right) which expresses how to handle luggage when there is a check-in of an unaccompanied minor traveling from EU to EU in economy class with a family member accompanying him to the boarding gate.

Figure 4: Replacements example for the “Check-in” family

Figure 5: VSpec model for the “Check-in” family

With both the base model and the replacement model and the VSpec model, we define the BVR realization model, also depicted in the upper right part of Figure 4. In this model we define the fragment substitutions, determining the correspondence between placements of the base model and replacements of the replacement model, within the VSpec model. We can notice the fragment substitution **V4Luggage** (colored), which replaces the **LuggagePlacement** placement defined in the base model (containing the **VariantLuggage** task) with the **V4Luggage** replacement (DLCR model) of the replacement model.

Each fragment substitution (variation point) corresponds to a VSpec variable that indicates when it should be executed. In the example, the replacement takes place when executing the transformation and the value of the DLCR VSpec (a choice) evaluates true. In the VSpec model depicted in Figure 5, we define the

Besides, for each fragment substitution the bindings must be defined between the elements of the substitution, as depicted in Figure 6. The bindings can be graphically defined by selecting the elements of the BPMN 2.0 process models. The selected fragments are highlighted within the process models, as can be seen in the upper left part of Figure 4, this example shows **VariantLuggage** which matches the **LuggagePlacement** placement in the base process with the **V4Luggage** replacement in the DLCR model.

BVR CheckIn3.bvr (Realization) ☰		
Variation points Bindings		
Type	[Object].property	Values
ToBinding	((SequenceFlow) baseFlow7).<FlowNode>targetRef	((SequenceFlow) DLCR_1).<FlowNode>targetRef[1]=[DLCR_PG_1]
ToBinding	((SequenceFlow) baseFlow8).<FlowNode>sourceRef	((SequenceFlow) DLCR_6).<FlowNode>sourceRef[1]=[DLCR_PG_2]
ToBinding	((Process) Default Process).<FlowElement>flowElements	((Process) DLCR Process).<FlowElement>flowElements[8]=[DLCR_3,DLCR_4,DLCR_PG_1,DLCR_5,Print duplicated b...
FromBinding	((ParallelGateway) DLCR_PG_1).<SequenceFlow>incoming	((Task) VariantLuggage).<SequenceFlow>incoming[1]=(baseFlow7)
FromBinding	((ParallelGateway) DLCR_PG_2).<SequenceFlow>outgoing	((Task) VariantLuggage).<SequenceFlow>outgoing[1]=(baseFlow8)

Figure 6: Binding example for the “Check-in” family

Finally, we need to express resolution models (one for each process variant). In Figure 7 we can see the BVR resolution model that corresponds to the variant we are working with as an example. In particular, the DLCR VSpec is set to true. The result of executing the transformation is the model depicted in Figure 8.

Figure 7: Resolution model for the “Check-in” family

Figure 8: Generated process variant for the “Check-in” family

We can see there that the DLCR variant was set in the process variant after the **PrintBoarding** task, since the **VariantPayment** was omitted (the **NotPay** VSpec within the resolution model depicted in Figure 7 is set to true). To get the other process variants, it is only required to define other resolution models while keeping the rest of the models unchanged.

4.2 Assignment of positions in academic exchange program process family

As presented in [16], the initial project lasted from 2013 to 2015, where, in the second phase of modeling, we worked with the central unit General Direction of Cooperation and International Relations (Dirección General de Cooperación y Relaciones Internacionales, DGRC³) on six selected BPs regarding academic exchange programs for graduate and postgraduate students and teachers, and collaboration projects between the university and external institutions (national and international).

By applying the methodology defined in [33] we identified and modeled a BP family that includes several variants for the exchange programs and collaboration projects. As a result, the BP family was modeled in BPMN 2.0 but based on using XOR gateways to check for every path what option should be taken regarding the type of exchange program or collaboration project (see [16] Fig.4 page 62). This type of variability modeling has several drawbacks, mainly adding complexity to the base process by including the variability directly on its control flow.

To improve this model and research on new proposals for modeling variability with other options over BPMN 2.0, we continued working on a second project, a research project funded by the Sectorial Commission of Scientific Research (Comisión Sectorial de Investigación Científica, CSIC⁴) from our University, from 2016 to 2019. The case study we present here (and the one presented in [10]) was carried out within this second project to validate our new proposals for modeling variability in BP families. The complete case study was then carried out in two parts:

- Part 1: the first part of the case study was carried out within the first project from 2013 to 2015, in which we identified and defined a process map for all management support BPs of the university, and modeling selected ones from the DGRC organizational unit, with BPMN 2.0 and the native support for variability it provides (based on XOR gateways), implementing a prototype within our BPM laboratory. The complete experience is described in [16] so we will not repeat it here.
- Part 2: the second part of the case study was carried out within the second project which lasted from 2016 to 2019, as we realized that the variability modeling obtained in Part 1 was complex and difficult for the stakeholders to manage, to improve the model and research on new proposals to model variability over BPMN 2.0. In this project, we investigated several existing variability approaches and defined two new ones over BPMN 2.0 (BPMNExt described in [10] and the one presented here) which were applied to the already modeled BP family for the exchange programs and collaborative projects of the DGRC, within our BPM laboratory.

In the following, we present the case study design, case selection, roles and procedures, application of the proposal, data collection and analysis, and validity threats for Part 2 of the case study.

4.2.1 Case study design

To present the case study we followed the guidelines and protocol definition and templates from [34, 35, 36]. We carried out a holistic single-case study in a single organization and within a single project, where the object of study was the application of our BPMN 2.0 process variability approaches (BPMNExt and the model-driven BVR/CVL approach), in the context of our university and BPM laboratory, being the unit of analysis the DGRC organizational unit of our university.

The main research question (RQ) for the case study was defined as: Does the variability approaches BPMNExt and model-driven with CVL defined over BPMN 2.0 provide appropriate and useful support for modeling BP families with BPMN 2.0 and generation of variants to be implemented in organizations?

For the part 2 of the case study we are presenting here, we focus on the model-driven approach with BVR/CVL over BPMN 2.0, asking specific questions on its use and results, as well as the tool support we provide, as follows: (RQ1) does the proposed approach provide appropriate and useful support for modeling BP families specified in BPMN 2.0 including variability points and corresponding variant parts, for process designers (technical and non-technical people)?; and (RQ2) is the tool support provided for modeling BP families and automated generation of corresponding variants appropriate and useful for process designers (technical and non-technical people)?

Apart from answering the specific research questions, we also assess our proposal through the VIVACE framework, not only by modeling the same process (cf. Section 4.1) but also by performing the evaluation of key characteristics for variability approaches defined by the framework (cf. Section 5) on our approach, which helps us compare our proposal to existing ones and evaluate its contributions.

³<https://cooperacion.udelar.edu.uy/>

⁴<https://www.csic.edu.uy/>

4.2.2 Case selection, roles, and procedures

As case, we selected the “Assignment of positions in academic exchange program” process family which we were working with from the first project, and as described in [16], was selected based on several characteristics it presents which makes it a representative one for the management area of the university. The criteria we used for the selection refers to: i) users, covering the largest number of participants within the university i.e. teachers, staff, students, other institutions, among others; ii) impact, involving several organizational units of the university; iii) Information Systems (IS) support, not having or having minimal support from traditional IS i.e. not focused on BPs; iv) definition, being already identified and availability of the corresponding organizational unit to further detail them, iv) the number of BP instances (cases), having a high number of cases, representing how much the BP is used within the university.

The BP family describes the process that is followed by our university to assign positions in academic exchange programs, in which, for every exchange program in which the university participates (Erasmus, Santander, Fulbright, etc.), positions are announced and interested people (students, faculty, staff) apply for them. Authorities from the DGRC organizational unit, which is in charge of the participation of the university in those exchange programs, and other authorities, analyze applicant’s merits and after some steps and meetings define the assignments, which are then notified to the applicants. For each exchange program, there is a variant describing which activities from the base process are carried out, depending on the normative of the program. Then, for each variation point, it can be performed or not, depending on the steps that the program requires. As mentioned before, from part 1 of the case study we already had the BP family modeled in BPMN 2.0 using XOR for deciding on performing each activity of the control flow, and a prototype implemented within two BPMS platforms.

We present here the generation of the variant for the Santander exchange program, in which the step **Evaluate Candidates** is deleted and the others steps are replaced by a simple task unless the **Assign Positions** step which is replaced by a sub-process to get applicants confirmation (holder or substitute in case the holder does not confirm) on taking the position until all positions are filled. Another variant (which we will not present here) corresponds to the PIMA exchange program, where all steps and meetings following the evaluation of the applicant’s merits are not carried out, and the list of applicants is prioritized and published directly. Both variants were modeled and generated with our BPMN 2.0 variability extension BPMNExt in [10].

The main roles that participated in the case study, as presented in [16], were the DGRC organizational unit of our university, and a working team on BPM including the first two authors of this paper, a researcher from the Faculty of Economics and Business Administration (Facultad de Ciencias Económicas y Administración), a software professional from the Central Computing Service of our University (Servicio Central de Informática UdelAR, SeCIU), and two members from the management team of the main sponsor of the project (Pro Rectorado de Gestión Administrativa). The third author participated in the second project with a focus on the proposal we are presenting here, performing the role of process designer.

The procedures for applying our proposal to the BP family were pretty straightforward since we already had the BPMN 2.0 model we defined in Part 1 with the DGRC as a basis. We performed the modeling of the BP family using the CVL approach (cf. Section 4.2.3), defining the base process, variability points, and corresponding variants, as well as the rest of the models that allow us to generate automatically the example process variant for the Santander exchange program. The process designer recorded the execution of the case study,

4.2.3 Application of the proposal

As mentioned before, to be able to model and automatically generate the desired variant within the CVL/BVR approach, we need to specify several models, starting with the base process for the BP family which is presented in Figure 9. Some activities, e.g., **Receive Applications** and **Control Documentation** are common for every process variant, whilst others, e.g., **Evaluate Candidates**, **Prioritize Candidates**, **Analyze Budget Availability**, and **Process Additional Documentation**, represent variation points. As before, all activities marked as a variation point in the base process, have variants associated with it, so we can replace it with some of the defined variants.

As explained in the first example, because of the modeling limitations of our solution, two artifacts that share different fragment substitution cannot be contiguous in the base model. Due to this, between the variations points **VP_Analyze Budget Availability**, **VP_Evaluate Candidates** and **VP_Prioritize Candidates** there are two intermediate events; the same happens between **VP_Negotiate Open Positions** and **VP_Assign Positions**. As mentioned, when the transformation is executed and the variant is generated, this auxiliary artifact is eliminated.

As before, the BPMN replacement processes for each variation point are modeled as independent pro-

Figure 9: Base process for the “Assignment of positions in academic exchange program” family

cesses, as shown in Figure 10. It can be seen that for each variation point there is a variant defined consisting of a task or a sub-process (in the case of the *VP_Assign Positions*), and the deletion operation where an intermediate element is modeled indicating the empty variant. This is modeled in this way since for each variation point there is only one possible variant defined when it occurs, but as presented in Section 4.1 it could also be the case that for a variation point, two or more possible different variants were defined to be used when it occurs within a specific process variant being defined, apart from the empty variant.

Figure 10: Replacements example for the “Assignment of positions in academic exchange program” family

The correspondence between a variation point and the corresponding variants shown in Figure 10 is:

- a) *VP_Analyze Budget Availability*: a.1) task *Analyze Budget Availability*, a.2) empty variant
- b) *VP_Evaluate Candidates*: b.1) task *Evaluate Candidates*, b.2) empty variant
- c) *VP_Prioritize Candidates*: c.1) task *Prioritize Candidates*, c.2) empty variant
- d) *VP_Negotiate Open Positions*: d.1) task *Negotiate Open Positions*, d.2) empty variant
- e) *VP_Assign Positions*: e.1) sub-process *Assign Positions*, e.2) empty variant
- f) *VP_Process Additional Documentation*: f.1) task *Process Additional Documentation*, f.2) empty variant

In the VSpec model, the defined variants for each variation point are associated with abstract variation points defined in the tree-like structure (feature model), as shown in Figure 11. It can be seen that for each abstract variation point the corresponding variant is presented (task or sub-process), and the empty variant.

Figure 11: VSpec model for the “Assignment of positions in academic exchange program” family

The realization model contains the mapping between the abstract variation points within the VSpec and the elements of the base model i.e. placement and replacement fragments, as shown in Figure 12, through defining fragment substitutions.

Figure 12 presents four quadrants: a) the variation point in the base process i.e. placement fragment; b) the corresponding variants i.e. replacement fragments; c) the fragment substitutions definition, and d) the placement and replacement fragment selected for the fragment substitution. Each fragment substitution that is defined in quadrant c) (tab variation points) corresponds to a VSpec variable that indicates when it should be executed.

In the example, we show the definition of the **Assign Positions** variation point (VP_Assign Positions) and the variant in which the sub-process **Assign Positions** defined will replace it, i.e. the “AM-Yes” option in the VSpec model.

Figure 12: BVR Realization model for the “Assignment of positions in academic exchange program” family

For each fragment substitution, the bindings must be defined between the elements of the substitution, as depicted in Figure 13. The bindings do not need to be defined textually, but graphically by selecting the elements of the BPMN 2.0 process models.

Then, the resolution model for the Santander process variant is defined, as shown in Figure 14, indicating for each option related with each variation point defined in the base process, whether it occurs or not within the variant. In particular, in this case, all **Yes** variants for variation points are set to **true** unless the one corresponding to the variation point **VP_EvaluateCandidates**, which has the **No** option set to **true**.

Finally, to generate the desired process variant an option is provided in the menu BPMN Variability for the selection of the required models, as shown in Figure 15. In particular, the resolution model (Configuration),

*santander.bvr (Realization)		
Variation points Bindings		
Type	[Object].property	Values
ToBinding	((SequenceFlow) sf11).<FlowNode>targetRef	((SequenceFlow) null).<FlowNode>targetRef[1] = {Assign Positions}
ToBinding	((SequenceFlow) sf12).<FlowNode>sourceRef	((SequenceFlow) null).<FlowNode>sourceRef[1] = {Assign Positions}
ToBinding	((BPMNShape) BPMNShape_Task_5).<BaseElement>bpmnElement	((BPMNShape) BPMNShape_SubProcess_2).<BaseElement>bpmnElement[1] = {Assign Positions}
ToBinding	((Process) Process).<FlowElement>flowElements	((Process) Pool 9 Process).<FlowElement>flowElements[1] = {Assign Positions}
FromBinding	((SubProcess) Assign Positions).<SequenceFlow>incoming	((Task) VP_Assign Positions).<SequenceFlow>incoming[1] = {sf11}
FromBinding	((SubProcess) Assign Positions).<SequenceFlow>outgoing	((Task) VP_Assign Positions).<SequenceFlow>outgoing[1] = {sf12}

Figure 13: Binding example for the “Assignment of positions in academic exchange program” family

Figure 14: Resolution model for the process variant “Santander”

the base process (Base), the variability model (Replacement), and the output file for the generation (Output).

Figure 15: Menu options for generating the process variant “Santander”

The “Santander” process variant is generated based on the definitions before and is shown in Figure 16. It can be seen that all artificial intermediate events in the base process were successfully eliminated, the variation points were replaced with the variants set to true (tasks and sub-process), and with the empty variant for the **Evaluate Candidates**. As for now, the type of the intermediate message in the sub-process replacing the **Assign Positions** variation point was manually set since messages types were not added yet in the BVR models.

4.2.4 Data collection and analysis

As mentioned before, data regarding the case study execution was recorded by the author playing the role of process designer, for each step of the process family modeling and generation of variants. The complete set of data is available in our repository [18] including the Eclipse project, the input and output files for the two assessments, the ATL transformation archive executed.

Regarding the experience as evaluated by the process designer, although the BVR/CVL approach requires several models to be defined to generate the desired variant, the step by step process was easily followed and

Figure 16: Generated process variant “Santander”

the variant successfully generated, as presented in Section 4.2. Also, having the process family modeled in BPMN 2.0 as input helped understand the family.

Regarding the BVR/CVL language, the fragment substitution that was included as a variability mechanism, is straightforward and clear in its use, including what to do and how to do it (based on the models defined). As for the tool support, it integrated the BVR tool and the BPMN 2.0 modeler in the Eclipse environment, which is a comfortable option mostly for technical people used to this type of platform. One element that should be improved in the future, is the fact that intermediate events have to be used when two fragment substitutions are contiguous in the base process. This adds unnecessary complexity to the base process of the family and obscures the family understanding.

Several internal documents were generated for each of the projects we carried out, as part of the complete case study, including process models, the prototype implementation, and several reports, mainly in Spanish for the university counterparts. Regarding the second part we presented here, we released several documents in Spanish to university authorities, including process variability material regarding the approaches we developed, which was also delivered to professionals and advanced students within a process variability school⁵ we carried out. Also, the screenshots of the step by step execution carried out can be accessed in the repository, where we also included the reference example described in Section 4.1 for the sake of completeness.

As the main results of the evaluation, we can argue that is feasible to model a process family with our BVR/CVL approach over BPMN 2.0 and generate automatically the desired process variant. We can also state that for process designers (technical people) the approach provides useful support for carrying out the modeling of process families (RQ1) and that the tool support we provide for modeling and generation of variants is also appropriate (RQ2).

4.2.5 Validity threats

There are validity threats that could affect the case study, which we describe in the following:

Construct validity:

- we defined the characteristics of the process family selected for the case study based on several characteristics that make the process a representative of processes carried out within a large organization as is our university.

⁵<https://www.fing.edu.uy/inco/eventos/bpmuy/>

External validity:

- the organization in which we carried out the case study presents several characteristics of organizations that would be interested in using the BVR/CVL model-driven approach: e.g. deals with a great number of BPs that are spread within different organizational units (not only management BPs but also others regarding teaching and researching) and differently defined and performed, mostly seen as different BPs instead of a process family.
- the first part of the case study apart from technical people also involved business people from the DGRC organizational unit, and the second part of the case study we presented here, dealt only with technical process designers. Although this fact can be seen as a threat, we can envision that non-technical process designers can deal with the proposal since the base modeling language (BPMN 2.0) is already known for them. Nevertheless, we need to carry out more executions to be able to generalize the results.

Reliability:

- we clearly defined the step by step execution of the case study to collect data from it within its execution, which we provided here for replication.

4.3 Discussion

We have presented two applications of the model-driven BVR/CVL proposal showing the applicability and results to automatically generate variants from a base process in a process family.

Although the first application, the “Airport Check-in” process family, is a very simple one, with a control flow that only contains sequential activities (and variation points), it has been used in the process family literature for years, providing a good reference for comparison between existing approaches. Also, the variants associated with the variation points contain several and different options, including the deletion option, and adding a fragment with gateways, which allows us to show the complexity of defining replacements and configuring the variants for the generation.

Regarding the second application, the case study on the “Assignment of positions in the academic exchange program” process family, not only is a real process family that we modeled out from our university as part of a larger case study that we added for the extension of this article, but it also presents a more complex control flow including several gateways between activities and variation points. On the other hand, the variants defined here for the variation points are simpler, since only contain two options: the defined variant or the delete option (empty variant), allowing us to show another type of configuration. The model of the process family used within this case study is a real process from the DGRC organizational unit of our university, which we modeled with businesspeople using BPMN 2.0 in the first part of the case study [16], using existing tools.

Based on the case study results, as presented in the analysis section (cf. Section 4.2.4) we can draw some conclusions about the proposal and its implications for the research community and the professional practice. The first one is regarding the feasibility of the approach for modeling BP process families and automatic generating the selected process variant, which was validated. Secondly, we can also affirm that the tool support is validated, as it allows performing the activities required to carry out the case study.

We claim that the proposal is appropriate and useful for technical process designers that deal with the definition and specification of BP process families, and the configuration of specific variants. Based on the availability of the proposal in our open repository [18] including the code, Eclipse project, input, and output files of the execution and screenshots, it can be used as-is, following the instructions we provide. We believe this is a major advantage of our proposal over other proposals since many of them do not provide tool support or it is not available due to being proprietary or discontinued or not easily integrated with other tools that are commonly used by business people and professionals.

Also, within this case study, we can compare the approach with the BPMN 2.0 variability extension BPMNExt we presented in [10] (cf. Section 4.3 for its presentation and analysis), for the modeling of the process family and the generation of variants. It can be seen that the base process where variation points are marked and common activities are defined, looks almost the same in the two approaches. We can argue that the modeling effort is comparable between the two proposals for the base processes. Nevertheless, from a more technical point of view of the modeling language, the BPMNExt is an extension of BPMN 2.0 that adds elements to it, while the BVR/CVL approach adds variability to BPMN 2.0 without adding elements to it since it adds variability over any language without changing the language.

As for the definition of variants and their association with variation points, both approaches present a configuration in which variation points and variants must be related so they can be selected. While this

is enough for defining the configuration of each process variant (i.e. which variant is to be used in which variation point) in the BPMNExt approach, in the CVL/BVR we also need to define three other models (i.e. VSpec model, BVR realization model, and BVR resolution model).

But, to generate the variant, when the five models contain the information required, it can be automatically generated with no other step, by using the transformations defined in our model-driven BVR/CVL approach. An important element is that the first four defined models can be reused as the basis for the generation of every process variant, and only the final model, the resolution model is required to be updated by changing the options selected for the generation of the new variant. In the BPMNExt approach, to generate the desired variant, each option for each variation point must also be selected to define the configuration but using the wizard we provide, and the generation is done programmatically. So it is fair to conclude that one major advantage of the BVR/CVL approach refers to the generation of specific process variants, which is facilitated by the model-driven approach.

We are working on an extended comparison between the three BP family models we worked with: i) BPMN 2.0 modeled with XOR; ii) BPMNExt approach; and iii) model-driven BVR/CVL approach, to assess the advantages and disadvantages that each option presents. The focus will be on process designers (technical and non-technical) that have to deal with modeling BPs families and generate the specific variants as required by organizations, to be implemented within the organization or different branch offices, even within different countries.

5 Assessment Using the VIVACE Framework

The VIVACE framework [17] was defined for the systematic assessment and comparison of process variability approaches "with respect to their expressiveness and the features provided for the support of process variability in the different phases of the process lifecycle". For doing so, VIVACE defines several characteristics for process variability support, which includes: i) process modeling language; ii) techniques for building configurable process models; iii) method for modeling the process family; iv) process perspectives covered; v) language constructs provided; vi) features provided to support process variability throughout the process lifecycle; vii) tool support; viii) empirical evaluation; and ix) application domain. All these elements must be assessed over the variability approaches under evaluation, providing the results in a table format that the framework defines. For the characteristics v) and vi) VIVACE defines a scale for the support provided: no support [-], partial support [+/-], and full support [+]. For the characteristic iv) it provides codes to indicate the perspectives covered: behavioral (B), functional (F), organizational (O), informational (I).

To illustrate the applicability of the VIVACE framework in [17], examples of assessment of selected process variability approaches are presented, for: i) C-EPC [37]; ii) Provop [25]; and iii) PESOA [28]. To the best of our knowledge, VIVACE was not previously used for the evaluation of CVL/BVR. Nevertheless, in this evaluation, we consider not CVL/BVR alone, but its adaptation to BPMN 2.0 for expressing business process families, as defined in previous sections. Table 2 provides a summary of compliance with each of the features proposed in the framework. The process we followed for the evaluation with VIVACE of the BVR/CVL model-driven approach is the one defined in [17] and carried out for the three approaches assessed by the authors, which includes for each of the characteristics defined by the framework, providing discussion and evidence of the evaluation results we informed, as we present in next sub-sections.

In Section 5.1 we present the evaluation for atomic characteristics (i.e. those that does not present nested characteristics), in Section 5.2 for v) language constructs provided and in Section 5.3 for vi) features provided to support process variability.

5.1 Atomic characteristics

In general terms, the approach uses a **multi-artifact** technique, based on an **independent** language for representing process variability (VSpec model, resolution model, etc., as introduced in Section 3.1) together with a particular business process modeling language (BPMN 2.0) for expressing the base model and the variants. Its method for modeling the process family is **operational** since there are concrete rules for adapting the base model (each VSpec kind) concerning an application context (the VSpec model), determining when these rules apply.

Theoretically, CVL/BVR covers **every process perspective** defined by VIVACE: Functional (F), change of a process activity (task or sub-process); Behavior (B), change of the dynamic behavior of a process (the control flow between its activities); Information (I), change of the informational entities consumed or produced during process execution; Organizational (O), change of the actors or roles involved in a process. However, since our adaptation to BPMN 2.0 is focused on the exploitation of a fragment substitution, thus is more concerned with variability concerning the functional perspective. Nevertheless, fragment substitution could also be used for expressing variability concerning the behavior, information, and organizational

Table 2: VIVACE-based evaluation of our approach

VIVACE		CVL / BVR for BPMN 2.0
Modeling language used to represent process variability		Independent
Technique used for building the configurable process model		Multi-artifact
Method for modeling the process family		Operational
Process perspectives covered		F, B, O, I
Variability-specific language constructs	LC1 Configurable Region	+ (FragmentSustitution - Placements)
	LC2 Configuration Alternative	+ (FragmentSustitution - Replacements)
	LC3 Configuration Context Condition	+ (VSpec)
	LC4 Configuration Constraint	+ (VSpec)
	LC5 Configurable Region Resolution Time	-
Variability support features	Analysis & Design phase	
	F1.1 Modeling a configurable process model	+ (BVR Tool - BPMN2ModelerAdapter)
	F1.2 Verifying a configurable process model and its related process family	-
	F1.3 Validating a configurable process model	-
	F1.4 Evaluating the similarity of different process variants	-
	F1.5 Merging process variants	+ (Files)
	Configuration phase	
	F2 Configuring specific regions of a process variant out of a configurable process model	+ (BVR Tool - fragmentSustitution)
	Enactment phase	
	F3.1 Configuring specific regions of a process variant at enactment time	-
	F3.2 Dynamically re-configuring an instance of a process variant at enactment time	-
	Diagnosis	
	F4 Analyzing a collection of process variants	-
	Evolution	
	F5.1 Versioning of configurable process model	+ (File versioning)
	F5.2 Propagating changes of a configurable process model to already configured process variants	-
Tool implementation		+ (BVR Tool - BPMN2Modeler - ATL)
Empirical evaluation		+
Application domain		education

perspectives, e.g., it is possible to change an activity consuming a data object and performed by a certain role, for parallel execution of two different activities producing other data objects and performed by certain other roles. An example of this is the introduction of a parallel behavior with the DLCR variant within the process family introduced in Section 4.1.

Regarding the tool support the approach provides a **complete tool** integrating the BVR Tool with fragment substitution, the BPMN 2.0 modeler, and the generation with ATL within the Eclipse environment, which is available for downloading in our repository [18]. The approach has been evaluated within a **case study** for a process family in the context of our university, which corresponds to the **education** domain.

5.2 Variability-specific language constructs

Configurable regions (LC1) are defined using the Placements of a Fragment Substitution which determines which elements (regions) in the base model can be substituted. For the configuration of alternatives (LC2), the Replacement of a Fragment Substitution is used, which defines which elements of the replacement model can be added to the base model in substitution of the associated placement. In our tool, placement and replacement fragments are highlighted within their corresponding models. The definition of which context depends on a variant (LC3) is made by matching fragment substitutions with nodes of the VSpec tree, that is, if the node is evaluated true, then the fragment substitution must be resolved. The VSpec model can be supplemented with constraints (LC4) related to VSpec elements, that must be taken into account for variant resolution. Finally, the language does not have any representation or primitive to determine the time of resolution of a configurable region (LC5).

5.3 Variability support features

5.3.1 Analysis & Design phase

Our integrated tool (BVR + BPMN 2.0) provides graphic support for the definition of a process family (F1.1). The tool indirectly supports the reuse of models (F1.5) since both the Placements and the Replacements of a BVR model contain references to the objects of the process models that are defined in separate files. However,

neither the tool nor the general approach provides mechanisms for proving correctness (F1.2), knowing the coverage (F1.3), or evaluating similarities (F1.4) for the generated variants. Concerning proving correctness, it is possible to use a BPMN 2.0 validation tool to verify if there is any variant that is not syntactically correct but after generating such variant. In the case of evaluating similarities, our work could be extended for generating all the possible variants and then using similarity measures, such as the ones proposed in [38], to be able to notify the user which variants cause a similar output.

5.3.2 Configuration phase

Besides CVL/BVR requires by definition that the specification must be executable, and the BVR tool already provides a code-based solution for the automatic generation of variants (F2), our integrated tool also provides a model-driven approach for carrying out the configuration process. This provides a higher-level representation of the process as well as a modular and reusable component.

5.3.3 Execution phase & Diagnosis

Neither the tool nor the general approach provide mechanisms for configuring specific regions of a process variant (F3.1), or dynamically re-configuring an instance of a process variant (F3.2), at enactment time. The tool does not support the analysis of process variants (F4) either.

5.3.4 Evolution

Since every model used within the approach is based on MOF, which has a standard representation in XML files, it is possible to use a software configuration management tool to keep the different versions of a process family (F5.1). Nevertheless, this feature is not currently supported within our tool. Finally, a mechanism for propagating changes is not provided (F5.2).

6 Other CVL Variation Points

As introduced in Section 3.1, we have used Fragment Substitution as the way of expressing a variation point, i.e., a specification of concrete variability in the base model. However, as shown in Figure 17, there are other four concrete kinds of variation points. In what follows we study how they can be integrated within our proposal, improving the coverage of the process perspectives defined by VIVACE (presented in Section 5).

Figure 17: Variation Point taxonomy from [12]

The **Slot Assignment** variation point is also defined in the original CVL specification (known as Value Assignment). It is a choice variation point (like the Fragment Substitution), i.e., the resolution of a choice determines whether or not the variation point will be applied or not. In this case, it specifies a value to be assigned to a particular slot (e.g., an attribute) in a particular object in the base model. Another version is the **Parametric Slot Assignment** which depends on a parameter for providing the value that will be

assigned to a particular slot. It can be used to change the concrete values of the properties of a BPMN 2.0 element without having to express a fragment. It could cover all process perspectives defined by VIVACE:

- Functional (F): change the type of an activity, e.g., from a user task to a service task;
- Behavior (B): change the type of a gateway, e.g., from an exclusive gateway (XOR) to an inclusive gateway (AND), type of events or the configuration of such gateways and events, e.g., changing the time of a timer or the condition of an exclusive gateway;
- Information (I): change the type of data, or data itself, related to an activity;
- Organizational (O): change the actors or roles that bind to a user task.

Take as an example the **Receive Applications** task of the case study base process depicted in Figure 9. We can use a slot assignment variation point related to the **timeDuration** attribute of its timer boundary event. With this, we can set a different time duration, depending on the kind of exchange program. We can define another variation point related with the **messageRef** attribute of the **Receive Applications** message task. In this case, we can vary the content and receiver of the message sent within different programs.

The **Staged Variation Point** does not exist in the original CVL specification. It refers to occurrences of VTypes (either VClassOccurrence or ChoiceOccurrence). The point is that realizing occurrences may vary from occurrence to occurrence of the very same VType. Typically the different occurrences of a VType correspond to desired changes in different concrete places of the base model. The semantics of a staged variation point is to limit the universe of variation points from which to choose. The VSpec being materialized is an occurrence that refers to a VType. That VType has a definition containing a tree of VSpecs. The resolution element triggering the staged variation point has a subtree of resolution elements that can only trigger variation points contained in the staged variation point. In the context of BPMN 2.0, the use of a staged variation point does not have a specific application beyond what has been said. However, it could be used for the definition of multi-level variation points, as done with BPMNNext [10], i.e., where the variants may contain variation points within. In this sense, the use of a staged variation point could allow defining the relationship between the selection of a specific variant and the options existing at the variation points defined in it. This aspect requires a more detailed study to analyze its viability.

Finally, the **Opaque Variation Point** is also defined in the original CVL specification. It is an executable, domain-specific variation point whose semantics is not defined by BVR. Instead, its semantics is defined based on a model transformation given as a textual definition in the given transformation language (e.g., ATL), that will be executed upon a positive decision of a Choice. With this kind of variation point we can implement complex variation points, e.g., defining variation points in terms of workflow patterns [39], or predefined change operations (insert, delete) as in the Provop [29] approach. As an example of the first, we can define an opaque variation point matching any Exclusive Choice (XOR-split gateway) and generating as an output an OR gateway adding a default option for notification. In the example, this pattern will match the XOR gateways which control assignments, and will: (a) change the gateway types to an OR-type; (b) add another sequence flow from the gateway to a send message task providing information about the remaining assignments. As an example of the second, we can define a delete operation, which takes any task with a simple input and a simple output, and deletes it, i.e., removes the task from the resulting process model and connects the source of its input flow with the target of its output flow. In the example, this can be used for deleting some of the variable tasks (e.g., Analyze Budget Availability or Prioritize Candidates depicted in Figure 9) instead of having empty replacement elements as those depicted in Figure 10.

7 Conclusions and Future Work

In this paper, we have explored an MDE-based strategy for modeling and configuring business process families. Our variability by extension approach is based on the CVL/BVR and BPMN 2.0 languages. These ideas were integrated into an Eclipse-based editor which allows modeling the process family with the common and variable parts, defining a concrete configuration, and automatically producing a concrete member of the process family i.e. a specific process variant (a resolved model).

We use BPMN 2.0 for expressing business process elements, in particular the base components of a process family, and CVL/BVR for expressing the variability aspects of the family as well as how the common and variable parts are related and how they can be configured. In particular, we made use of a single kind of variation point: a Fragment Substitution, since it seemed to be the most appropriate to be applied in most cases of BPMN. It allows expressing that some parts of a common process model (a fragment) can be substituted by some variant (another fragment). We also study the application of other kinds of variation points which still have to be integrated within our solution. Moreover, as done in [27], it could be interesting

to study how to define nested substitutions (multi-level variability), i.e., a fragment could not only belong to a fragment substitution as a replacement (a variant) but also as a placement as part of another substitution (a variation point within a variant).

We also essayed a model-driven approach for carrying out the configuration of process variants from a process family in this context, by means of an ATL model transformation. It provides a higher-level representation of the steps involved in the configuration process and the constraints that must apply between the variants. It also provides a modular and reusable component that improves maintainability for the automated generation of process variants. Interesting future work is to study how to ensure the correctness of all the derivable variants before performing the transformation.

Moreover, we validate the proposal and the tool support by two applications of the approach: one taken from the variability literature and a case study from a real project we carried out within our university. While one of the advantages of CVL/BVR is that it is not necessary to add new elements to identify variation points in the base model, it is also somewhat complex to solve, although it is facilitated by graphically selecting elements using the BPMN2 Modeler we have developed. Further analysis and complete case studies are needed to comprehensively assess the expressiveness and understanding of the approach by real users.

Finally, we evaluate the approach using the VIVACE framework which provides common settings for a comparative evaluation of any process variability proposal with respect to others in the literature. By using this framework, we have detected many variability support features that could be of interest but still not supported by our proposal, e.g., the aforementioned correctness checking of the derivable process variants. A comparative evaluation of our proposal and other BPMN 2.0 based approaches is under development.

Acknowledgment

This work was partially funded by project “Generación automática de variantes en familias de procesos de negocio”, Comisión Sectorial de Investigación Científica (CSIC), Universidad de la República, Uruguay. We thank the undergraduate students Darwin Fernandez and Gonzalo López who worked on the development.

References

- [1] M. Weske, *BPM - Concepts, Languages, Architectures, 3rd Edition*. Springer, 2019.
- [2] W. M. P. van der Aalst, A. H. M. ter Hofstede, and M. Weske, “Business process management: A survey,” in *Business Process Management, International Conference, BPM 2003, Eindhoven, The Netherlands, June 26-27, 2003, Proceedings*, ser. Lecture Notes in Computer Science, W. M. P. van der Aalst, A. H. M. ter Hofstede, and M. Weske, Eds., vol. 2678. Springer, 2003, pp. 1–12.
- [3] M. Dumas, M. L. Rosa, J. Mendling, and H. A. Reijers, *Fundamentals of Business Process Management, Second Edition*. Springer, 2018.
- [4] M. Dumas, W. M. P. van der Aalst, and A. H. M. ter Hofstede, Eds., *Process-Aware Information Systems: Bridging People and Software Through Process Technology*. Wiley, 2005.
- [5] M. Reichert and B. Weber, *Enabling Flexibility in Process-Aware Information Systems - Challenges, Methods, Technologies*. Springer, 2012.
- [6] M. L. Rosa, W. M. P. van der Aalst, M. Dumas, and F. Milani, “Business process variability modeling: A survey,” *ACM Comput. Surv.*, vol. 50, no. 1, pp. 2:1–2:45, 2017.
- [7] OMG, “Business Process Model and notation (BPMN) v2.0,” Object Management Group (OMG), Tech. Rep., 2013.
- [8] G. Valenca, C. Alves, V. Alves, and N. Niu, “A systematic mapping study on business process variability,” *International Journal of Computer Science & Information Technology (IJCSIT)*, vol. 5, no. 1, pp. 1–21, 02 2013.
- [9] C. Ayora, V. Torres, V. Pelechano, and G. H. Alférez, “Applying CVL to business process variability management,” in *Proceedings of the VARIability for You Workshop - Variability Modeling Made Useful for Everyone, VARY '12, Innsbruck, Austria, September 30, 2012*, Ø. Haugen, J. Jézéquel, A. Wasowski, B. Møller-Pedersen, and K. Czarnecki, Eds. ACM, 2012, pp. 26–31.
- [10] A. Delgado and D. Calegari, “BPMN 2.0 based modeling and customization of variants in business process families,” in *2017 XLIII Latin American Computer Conference, CLEI 2017, Córdoba, Argentina, September 4-8, 2017*, H. Monteverde and R. Santos, Eds. IEEE, 2017, pp. 1–9.

- [11] OMG, “Common Variability Language (CVL). Revised Submission,” Object Management Group (OMG), Tech. Rep., 2012.
- [12] Ø. Haugen and O. Øgård, “BVR - better variability results,” in *System Analysis and Modeling: Models and Reusability - 8th International Conference, SAM 2014, Valencia, Spain, September 29-30, 2014. Proceedings*, ser. Lecture Notes in Computer Science, D. Amyot, P. F. i Casas, and G. Mussbacher, Eds., vol. 8769. Springer, 2014, pp. 1–15.
- [13] S. Kent, *Model-Driven Engineering*, ser. Lecture Notes in Computer Science. Springer, 2002, vol. 2335.
- [14] OMG, “Meta Object Facility (MOF) v2.5.1,” Object Management Group (OMG), Tech. Rep., 2016.
- [15] F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev, “ATL: A model transformation tool,” *Sci. Comput. Program.*, vol. 72, no. 1-2, pp. 31–39, 2008.
- [16] A. Delgado and D. Calegari, “Changing the focus of an organization: From information systems to process aware information systems,” in *Enterprise, Business-Process and Information Systems Modeling - 16th International Conference, BPMDS 2015, 20th International Conference, EMMSAD 2015, Held at CAiSE 2015, Stockholm, Sweden, June 8-9, 2015, Proceedings*, ser. Lecture Notes in Business Information Processing, K. Gaaloul, R. Schmidt, S. Nurcan, S. Guerreiro, and Q. Ma, Eds., vol. 214. Springer, 2015, pp. 53–67.
- [17] C. Ayora, V. Torres, B. Weber, M. Reichert, and V. Pelechano, “VIVACE: A framework for the systematic evaluation of variability support in process-aware information systems,” *Inf. Softw. Technol.*, vol. 57, pp. 248–276, 2015.
- [18] L. Peña, D. Fernandez, and G. López, “BVR+BPMN 2.0 Editor,” <https://gitlab.fing.edu.uy/open-coal/bpmn-cvl>, 2018.
- [19] D. Calegari, A. Delgado, and L. Peña, “Automated generation of variants in business process families based on the common variability language (CVL),” in *XLV Latin American Computing Conference, CLEI 2019, Panama, September 30 - October 4, 2019*. IEEE, 2019, pp. 1–10.
- [20] M. Terenciani, D. M. B. Paiva, G. Landre, and M. I. Cagnin, “Bpmn* - A notation for representation of variability in business process towards supporting business process line modeling,” in *The 27th International Conference on Software Engineering and Knowledge Engineering, SEKE 2015, Wyndham Pittsburgh University Center, Pittsburgh, PA, USA, July 6-8, 2015*, H. Xu, Ed. KSI Research Inc. and Knowledge Systems Institute Graduate School, 2015, pp. 227–230.
- [21] R. M. Pillat, T. C. Oliveira, P. S. C. Alencar, and D. D. Cowan, “Bpmnt: A BPMN extension for specifying software process tailoring,” *Inf. Softw. Technol.*, vol. 57, pp. 95–115, 2015.
- [22] H. Zhang, W. Han, and C. Ouyang, “Extending BPMN for configurable process modeling,” in *Moving Integrated Product Development to Service Clouds in the Global Economy - Proceedings of the 21st ISPE Inc. International Conference on Concurrent Engineering, Beijing Jiaotong University, China, September 8-11, 2014*, ser. Advances in Transdisciplinary Engineering, J. Cha, S. Chou, J. Stjepandic, R. Curran, and W. Xu, Eds., vol. 1. IOS Press, 2014, pp. 317–330.
- [23] V. Kulkarni and S. Barat, “Business process families using model-driven techniques,” *Int. J. Bus. Process. Integr. Manag.*, vol. 5, no. 3, pp. 204–217, 2011.
- [24] G. Gröner, M. Boskovic, F. S. Parreiras, and D. Gasevic, “Modeling and validation of business process families,” *Inf. Syst.*, vol. 38, no. 5, pp. 709–726, 2013.
- [25] A. Hallerbach, T. Bauer, and M. Reichert, “Capturing variability in business process models: the provop approach,” *J. Softw. Maintenance Res. Pract.*, vol. 22, no. 6-7, pp. 519–546, 2010.
- [26] M. Döhring and B. Zimmermann, “vbpmn: Event-aware workflow variants by weaving BPMN2 and business rules,” in *Enterprise, Business-Process and Information Systems Modeling - 12th International Conference, BPMDS 2011, Proceedings*, ser. Lecture Notes in Business Information Processing, vol. 81. Springer, 2011, pp. 332–341.
- [27] A. Delgado, D. Calegari, and F. García, “Modeling of software process families with automated generation of variants (S),” in *The 30th International Conference on Software Engineering and Knowledge Engineering, Hotel Pullman, Redwood City, California, USA, July 1-3, 2018*, Ó. M. Pereira, Ed. KSI Research Inc. and Knowledge Systems Institute Graduate School, 2018, pp. 330–329.

- [28] F. Puhlmann, A. Schnieders, J. Weiland, and M. Weske, “Variability mechanisms for process models,” PESOA-Report TR 17/2005, Process Family Engineering in Service-Oriented Applications (PESOA). BMBF-Project, Tech. Rep., 2005.
- [29] M. Reichert, S. Rechtenbach, A. Hallerbach, and T. Bauer, “Extending a business process modeling tool with process configuration facilities: The provop demonstrator,” in *Proceedings of the Business Process Management Demonstration Track (BPMDeMos 2009), Ulm, Germany, September 8, 2009*, ser. CEUR Workshop Proceedings, A. K. A. de Medeiros and B. Weber, Eds., vol. 489. CEUR-WS.org, 2009.
- [30] E. Santos, J. Pimentel, J. Castro, J. Sánchez, and O. Pastor, “Configuring the variability of business process models using non-functional requirements,” in *Enterprise, Business-Process and Information Systems Modeling - 11th International Workshop, BPMDS 2010, and 15th International Conference, EMMSAD 2010, held at CAiSE 2010, Hammamet, Tunisia, June 7-8, 2010. Proceedings*, ser. Lecture Notes in Business Information Processing, I. Bider, T. A. Halpin, J. Krogstie, S. Nurcan, E. Proper, R. Schmidt, and R. Ukör, Eds., vol. 50. Springer, 2010, pp. 274–286.
- [31] I. Machado, R. Bonifácio, V. Alves, L. Turnes, and G. Machado, “Managing variability in business processes: An aspect-oriented approach,” in *Proceedings of the 2011 International Workshop on Early Aspects*, ser. EA ’11. New York, NY, USA: Association for Computing Machinery, 2011, p. 25–30.
- [32] T. Nguyen, A. W. Colman, and J. Han, “Modeling and managing variability in process-based service compositions,” in *Service-Oriented Computing - 9th International Conference, ICSOC 2011, Paphos, Cyprus, December 5-8, 2011 Proceedings*, ser. Lecture Notes in Computer Science, G. Kappel, Z. Maa-mar, and H. R. M. Nezhad, Eds., vol. 7084. Springer, 2011, pp. 404–420.
- [33] A. Delgado, B. Weber, F. Ruiz, I. G. R. de Guzmán, and M. Piattini, “An integrated approach based on execution measures for the continuous improvement of business processes realized by services,” *Inf. Softw. Technol.*, vol. 56, no. 2, pp. 134–162, 2014.
- [34] P. Runeson and M. Höst, “Guidelines for conducting and reporting case study research in software engineering,” *Empir. Softw. Eng.*, vol. 14, no. 2, pp. 131–164, 2009.
- [35] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, and B. Regnell, *Experimentation in Software Engineering*. Springer, 2012.
- [36] R. K. Yin, *Case study research. Design and methods, 6th edition*. Sage Publications, 2017.
- [37] M. Rosemann and W. M. P. van der Aalst, “A configurable reference modelling language,” *Inf. Syst.*, vol. 32, no. 1, pp. 1–23, 2007.
- [38] M. Becker and R. Laue, “Analysing differences between business process similarity measures,” in *Business Process Management Workshops*, F. Daniel, K. Barkaoui, and S. Dustdar, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 39–49.
- [39] N. Russell, W. M. P. van der Aalst, and A. H. M. ter Hofstede, *Workflow Patterns: The Definitive Guide*. MIT Press, 2016.