


A Design Methodology for Modeling Flexible Business Processes

Imen Ben Said, University of Sfax, Tunisia*

Mohamed Amine Chaabane, MIRACL, ISAAS, University of Sfax, Tunisia

Rafik Bouaziz, MIRACL, FSEGS, University of Sfax, Tunisia

 <https://orcid.org/0000-0001-5398-462X>

ABSTRACT

Business process flexibility is a major challenge since it is crucial to consider the current environment, which is becoming very fluctuating, and to adapt processes accordingly. To deal with this issue, version-based approach is recognized as a key notion. This approach allows defining several versions for each process to take into account the significant changes occurring to this process while keeping track of updates. Several contributions have proposed solutions based on versions to deal with process flexibility. However, to the authors' knowledge, none of them offered a version derivation approach under a design methodology to help designers to model and handle versions of processes. So, this paper recommends a design methodology based on versions, including a derivation approach to create new versions based on previously modeled ones.

KEYWORDS

Alternative, Approach, BPMN, Business Process, Context Awareness, Derivation, Editor, Elolution, Flexibility, Methodology, Versions

1. INTRODUCTION

Nowadays organization's business environments become more and more complex, dynamic, and evolving due to internal and external factors such as economic changes, technologic innovations and sanitary conditions. So, disturbances can affect their business routines, requiring Business Processes (BP) capable of adapting to frequent changes. Thus, for surviving in such environment, organizations have to face an important issue in Business Process Management (BPM) area; it's about the Business Process Flexibility (Barba et al., 2021; Jemal et al., 2018; van Eck et al., 2015).

Business Process flexibility is defined as the ability of a business process to accommodate organizational, functional and/or operational changes occurring in its operating environment (Chaabane et al., 2010). In order to feature this requirement, several taxonomies have been proposed in the

DOI: 10.4018/ijismd.315025

*Corresponding Author

This article published as an Open Access article distributed under the terms of the Creative Commons Attribution License (<http://creativecommons.org/licenses/by/4.0/>) which permits unrestricted use, distribution, and production in any medium, provided the author of the original work and original publication source are properly credited.

literature. The more suitable one is given in (Reichert & Weber, 2012). This taxonomy highlights two different times when process flexibility is considered: flexibility at design-time, which refers to foreseeable changes that can be taken into account in modeled process schema, and flexibility at run-time, which refers to unforeseeable changes occurring during process executions. In addition, this taxonomy identifies four needs of flexibility: (i) variability, for representing a process differently, depending on the context of its execution, (ii) adaptation, for handling occasional situations or exceptions which have not been necessarily foreseen in process schema, (iii) evolution, for handling changes in processes, which require occasional or permanent modifications in their schema, and (iv) looseness, for handling processes whose schema are not known a priori and which correspond to non-repeatable, unpredictable, and emergent processes.

To address process flexibility in BPM, several contributions have recommended (i) *variant-based* approach (Aysolmaz et al., 2017; Hallerbach et al., 2010; Milani et al., 2016; Natschläger et al., 2016; Reichert et al., 2015; Rosa et al., 2009; Tealeb et al., 2014), which recommends reuses a set of variants of the process components (i.e., the process itself, its sub-processes and its tasks), each variant being convenient to a given context, and (ii) *version-based* approach (Chaâbane et al., 2010; Chaâbane et al., 2020; Ekanayake et al., 2011; Ellouze et al., 2017; Ellouze et al., 2016; Lassoued et al., 2016; Zhao & Liu, 2013), which is an extension of the variant-based as it allows using a set of alternative versions (equivalent to variants), on the one hand, and defining consecutive versions of the process (following evolutions of some versions), on the other hand. A version corresponds to one of the significant states that a process component (the process itself, or its sub-processes or tasks) may have during its life cycle (Chaâbane et al., 2010). In fact, evolution and versioning are related not only to the ability to perform updates on a process, a sub-process or a task model, but also to the ability to keep track of these updates. This is why we adopt the *version-based* approach. The latter consists in the specification of several schema versions of each process to take into account the significant changes occurred in that process and its components (i.e., activities, information and resources involved in its execution). Each new version must be defined by derivation from previous one forming a derivation hierarchy. The notion of version has been recognized as a key notion to deal with flexibility issue (Reichert & Weber, 2012).

However, designers also need an appropriate methodology to model business processes based on versions to better address flexibility. But Existing contributions recommending versioning for process flexibility have mainly focused on the first component of a design methodology, i.e., a language that defines a specific notation for expressing models. Some of them have introduced versioning models (Ekanayake et al., 2011) and graphs, e.g., Versioned Preserving directed Graph VPG (Zhao & Liu, 2013). Others have proposed specific meta-models supporting the modeling of versions (Chaâbane et al., 2010; Ellouze et al., 2016; Lassoued et al., 2016). Moreover, these contributions did not take into account the contextual dimension of processes in order to characterize the situations in which instances of processes are executed (Nurcan & Edme, 2005; Saidani et al., 2015). We believe it is important to compensate for these drawbacks.

So this paper provides a comprehensive methodology, with three components, for the design and implementation of versions of processes models. This methodology includes a derivation approach to help designers to manage versions supporting flexible processes. In the first component, we start by presenting the BPMN4V-context meta-model proposed in (Chaâbane et al., 2020) which is an extension of BPMN meta-model (OMG, 2014) integrating versions and their context. We then show how to use the notation (i.e., language) related to this meta-model to model versions and their contexts. The second component of this methodology is a build process composed of a set of steps that help BPM designers to derive new versions from existing ones. In fact, when there is a need of derivation, it is important to retrieve the most appropriate versions that can be the source of this derivation. The build process aims to help designers to efficiently perform version derivation. Finally, the third component provides a tool for modeling and deriving versions of process models, according to the BPMN4V-context notation and the build process.

Accordingly, this paper is organized as follows. Section 2 provides a study of related work: state-of-the-art for BP flexibility; variant and version based approaches. Section 3 introduces the fundamental ideas and the origin of the proposed design methodology: derivation approach and BPMN4V-context meta-model. As for section 4, it is dedicated to the three components of this methodology: notation to model versions and their contexts in accordance with BPMN4V-context meta-model; build process for building versions and deriving new versions from existing ones; design tool features. Section 5 presents a proof of concept based on a case study used to illustrate the paper contributions. Section 6 presents BPMN4V-context Architect, the tool implementing the proposed notation and the build process according to the derivation approach. Finally, Section 7 concludes the paper and gives some directions for future work.

2. RELATED WORKS

In this paper we deal with BP flexibility at design-time which consists on handling processes changes in accordance with context variation. Particularly, we present, in this section, some definitions related to context-awareness in BPM area literature, and we examine the most popular techniques addressing flexibility.

If some contributions addressed the modeling of versions in BPM, they considered especially the behavioral, organizational and informational dimensions, which respectively concern the definition of the activities of a process with their synchronization, the involved resources in the realization of these activities and the data they produce or consume. However, to address process modeling in a comprehensive way, it is not enough to just consider these three dimensions; it is necessary to also take into account the contextual dimension of processes in order to characterize the situations in which instances of processes are executed (Nurcan & Edme, 2005; Saidani et al., 2015).

Context-awareness has been deeply investigated in BPM area; according to (Rosemann & Recker, 2006), process context is defined as “the minimum set of elements containing all relevant information that impact the design and execution of a process”. To classify these context elements, four types of context are distinguished in (Rosemann et al., 2008): (i) the Immediate context, which covers elements on process components, namely context of activities, events and resources, (ii) the Internal context, which includes elements on the internal environment of an organization that impacts its processes, (iii) the External context, which encompass elements relating to external stakeholders of the organization, and (iv) the Environmental context, which contains elements related to external factors.

To address process flexibility in BPM, mainly two main types of approaches have been proposed in literature: the *variant-based* approach and the *version-based* approach. A variant is an adjustment of a basic process model to requirements, whether expected or unexpected, of process contexts. Therefore, several existing works (Aysolmaz et al., 2017; Hallerbach et al., 2010; Natschläger et al., 2016; Reichert et al., 2015; Rosa et al., 2009; Tealeb et al., 2014) around the variant technique propose a set of configurations to this basic process using adaptation patterns (Hallerbach et al., 2010). For instance, the *variant-based* approach recommended in (Rosa et al., 2009) allows the modeling of several variants for fragments or tasks, each variant being convenient to a given context. However, only one fragment schema is kept for each variant and there is no track of the different changes performed to each variant (none of these works supports traceability). In fact, flexibility is not related only to the ability to perform updates on process, sub-process or task schema, but also to the ability to keep track of these updates.

Other works have adopted *version-based* approach that consists on the specification of business process changes through different versions. A version corresponds to one of the significant states a process may have during its life cycle. Each new version must be defined by derivation from previous one forming a derivation hierarchy. The notion of version has been recognized as a key notion to deal with flexibility issue. First, using versions allows keeping track of process evolution. This ensures not only process traceability but also the storage of previously modeled versions in order to promote

their reuse in similar context. Second, versions are appropriate to deal with process evolution, process adaptation and process variability, which are the main process flexibility needs in BPM (Reichert & Weber, 2012). Finally, handling versions of process schema facilitates the migration of instances from an initial schema version to a final one, allowing if the migration is not possible, two different instances of a same process to run according to two different schema versions (Zhao & Liu, 2013). As consequence, we advocate in this paper a *version-based* approach to deal with business process flexibility.

To confirm this choice, let's present in what follows the most pertinent literature works that also recommend a version-based approach to deal with process flexibility. These works have addressed the modeling of processes in BPM in accordance with the *version-based* approach (e.g., (Chaâbane et al., 2010; Chaâbane et al., 2020; Ekanayake et al., 2011; Ellouze et al., 2017; Ellouze et al., 2016; Zhao & Liu, 2013)). In (Ekanayake et al., 2011) and (Zhao & Liu, 2013), authors have introduced specific versioning models that support the modeling of versions for concepts relevant to the behavioral dimension of processes only. As for (Chaâbane et al., 2010) and (Ellouze et al., 2016), authors have proposed their own meta-models to support versions modeling considering mainly the behavioral, informational and organizational dimensions, and partially the contextual dimension. Unfortunately, the proposed meta-models are not likely to be used by BPM practitioners since they are not based on standards. To overcome these drawbacks, the BPMN4V-context meta-model proposed in (Chaâbane et al., 2020) extends the BPMN2.0 standard to integrate (i) version modeling capability and (ii) contextual information for versions use; BPMN4V-context supports the definition of versions of processes as well as the context of each version.

We give in Table 1 an evaluation of the existing contributions in accordance with the versioning approach to consider flexible processes at design-time. This evaluation is based on the following criteria:

- Dimensions: this criterion specifies whether (or not) the examined work supports the behavioral, informational, organizational, and/or contextual dimensions in versions modeling.
- Standard: this criterion checks if the proposed solution is based on a standard or not.
- Derivation process: this criterion verifies whether (or not) the examined work proposes a set of steps for deducing new versions from existing ones.
- Implementation: this criterion specifies if there is a tool that implements the recommended solution or not.

Table 1. Comparison of existing works advocating version approach to deal with flexible processes

Works Criteria		Ekanayake et al. (2011)	Zhao and Liu (2013)	Chaâbane et al. (2010)	Ellouze et al. (2016)	Chaâbane et al. (2020)
Dimensions	behavioural	+	+	+	+	+
	informational	-	-	+	+	+
	organizational	-	-	+	+	+
	contextual	-	-	+/-	+/-	+
Standard		+	-	-	-	+
Derivation process		-	-	-	-	-
Implementation		-	-	-	-	-

Legend: (+) means that the feature is supported, (-) means that the feature is not supported, and (+/-) means that the feature is partially supported.

As indicated in Table 1, we note that the behavior dimension is evoked by all the examined works in order to capture changes related to a process, i.e., the activities and the events that make up this process, and the way of their coordination. The authors of (Chaâbane et al., 2010; Ellouze et al., 2016) also considered the informational and organizational dimensions to track changes related to the manipulated data and the involved resources. In addition, they partially addressed the contextual dimension; they did not consider all types of contextual information to describe process context in a comprehensive way. The authors of (Chaâbane et al., 2020) overcame these weaknesses; they have proposed an extension to BPMN notation to consider versions modeling with respect to the behavioral, informational, organizational and contextual dimensions. We believe that this contribution is interesting since BPMN is a standard promoted by the OMG and is more used by the BPM community than specific ad-hoc notations. However, in (Chaâbane et al., 2020) authors have just proposed the BPMN4V-context meta-model without defining how we can use it as part of a comprehensive design methodology. Moreover, some other contributions recommending versioning for process flexibility have mainly focused on the first components of a design methodology: introduction of versioning models (Ekanayake et al., 2011) and graphs, e.g., Versioned Preserving directed Graph VPG (Zhao & Liu, 2013). Others have proposed specific meta-models supporting the modeling of versions (Chaâbane et al., 2010; Ellouze et al., 2016; Lassoued et al., 2016).

The work presented in this paper aims at overcoming the drawbacks of existing works. It recommends a comprehensive design methodology, called METHOD4BPVC, allowing the support of flexible BP at design-time while advocating a derivation approach.

3. FUNDAMENTAL IDEAS AND ORIGIN OF THE METHODOLOGIE METHOD4BPVC

METHOD4BPVC is the design methodology we propose for modeling BP versions and their contexts. It is dedicated to help design versions of BP models using a derivation approach to create new versions based on previous modeled ones in accordance with the BPMN4V-Context meta-model we proposed in (Chaâbane et al., 2020) as an extension of the BPMN2.0 meta-model.

The methodology METHOD4BPVC is composed of three components as shown in Figure 1.

For the first component, we propose a notation (i.e., a design language) conforming to the BPMN4V-Context meta-model (Chaâbane et al., 2020). This notation, entitled BPMN4V-Context notation, is useful since it is based on BPMN standards, and it allows modeling versions of processes taking into account all process dimensions.

Regarding the second component, it provides a build process that helps BPM designers to efficiently derive versions of processes modeled according to BPMN4V-Context notation. This process is composed of five steps allowing version derivation.

Finally, the third component of this methodology is an appropriate tool which offers visual editor and wizards for modeling and managing versions with respect to the BPMN4V-Context notation and following the build process. This tool is called BPMN4V-Context Architect.

As the notation of METHOD4BPVC is to be defined in accordance with the BPMN4V-Context meta-model (Chaâbane et al., 2020) while including a derivation approach, we briefly present here after this approach and this meta-model.

3.1 Derivation approach:

METHOD4BPVC includes a derivation approach to support BP flexibility at design-time. This approach is based mainly on the notion of version and the notion of context. More precisely, it advocates that versions make it possible to follow the BP changes, on the one hand, and to define an appropriate version for each specific situation/context, on the other hand. Figure 2 illustrates versions of a process named P. These versions are linked by a derivation link; they form a derivation hierarchy. Two types of derivation have been proposed: alternative derivation, for creating versions that can

be alternatively used, and evolution derivation, for evolving the process model from one version to another. The definition of every new version is done by derivation from a previous one (except the first version of the process, which is created from scratch – e.g., P. V1): such versions are called derived versions. Of course, several versions may be derived from the same previous one: they can be alternative versions (e.g., P.V1.1, P.V1.2 ...) or evolution versions (e.g., P.V2).

Thus, we adopt the versioning pattern proposed in (Ben Said et al., 2016) (see Figure 3) making some classes of BPMN2.0 meta-model able to support version modeling. The underlying idea is to allow modeling, for each versionable concept of the BPMN meta-model, both entities and their corresponding versions. A versionable concept is a concept for which it is important to track its changes by handling versions. Each versionable concept is described as a class, called “*Versionable*”. A *Versionable* class is associated to a new class, called “*Version_of_Versionable*”, whose instances are versions of the *Versionable* class, and two relationships: (i) the “*is_version_of*” composition, which links each instance of the *Versionable* class with its corresponding instances of the *Version_of_Versionable* class, (ii) the “*derived-From*” relationship, which supports the version derivation hierarchy modeling. This latter relationship is reflexive; the semantics of both relationship sides are: a version (SV) succeeds another one in the derivation hierarchy and a version (PV) precedes another one in the derivation hierarchy. Moreover, this relationship includes the link attribute “*type*” used to specify the type of derivation (derivation for alternative or derivation for evolution).

3.2 BPMN4V-Context meta-model (Chaâbane et al., 2020):

BPMN4V-Context extends the BPMN2.0 meta-model by (i) a versioning pattern making some BPMN2.0 meta-model classes able to support version modeling and (ii) some context meta-classes for considering version contexts. The resulting meta-model, given in Figure 4, defines the necessary concepts for supporting versions of process models and their contexts. In this figure, white rectangles and links correspond to meta-classes and meta-relationships of BPMN2.0, grey rectangles and blue links correspond to meta-classes and meta-relationships involving versions following the introduction of the versioning pattern, and red and green rectangles and links represent meta-classes and meta-relationships used to support context.

More precisely, the versioning pattern allows defining versionable concepts (e.g., process, sub-process, activity, etc.) in the BPMN4V-Context meta-model, and storing both the entities and their corresponding versions (see right part of Figure 4). A versionable concept is a concept for which it is

Figure 1. Design methodology components

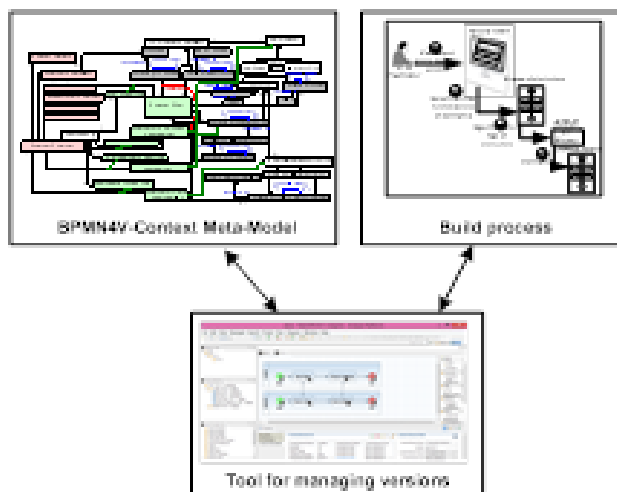


Figure 2. Versions to Model Process flexibility

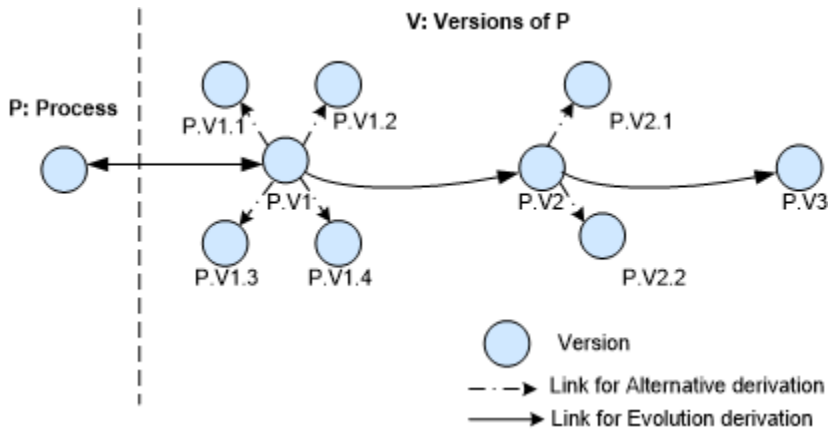
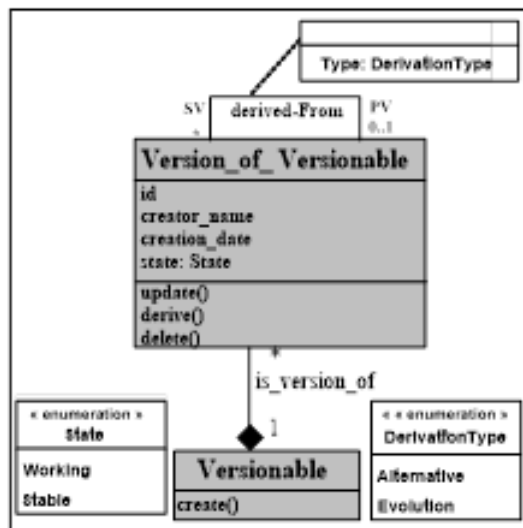


Figure 3. Versioning pattern



important to track its changes by handling versions. BPMN4V-Context provides seven meta-classes to deal with versions of concepts: *Process*, *Sub Process*, *Event*, *Task*, *ItemAwareElement*, *Resource*, and *ResourceRole*. Each *Versionable* meta-class (e.g., *Process*) is associated to a new meta-class whose instances are versions of this *Versionable* meta-class (e.g., *Version of Process*).

A new version of a versionable concept is defined according to changes occurring to it: these changes may correspond to the adding of new information (property or relationship) or to the modification or the deletion of an existing one. For instance, we create a new version of a process model when there are changes in the involved activities and events or in their coordination (using patterns for changes in gateways and sequence flow).

Regarding context modeling, the BPMN4V-Context meta-model includes the meta-class *Context Container* which defines an aggregation of a set of context elements (see left part of Figure 4). Each instance of the *Context Element* meta-class corresponds to a contextual information characterizing a situation, and to which a condition is defined. A context nature is associated to each context

element through the meta-class *Context Nature*. This nature can be immediate, internal, external, or environmental, according to the taxonomy given in (Rosemann & Recker, 2006). In addition, a context element refers to the dimension to which it belongs to, through the specialization of *Context Element*. Thus, an instance of *Context Element* can be an instance of: (i) *Goal Element*, i.e., an element describing the objective to be achieved (e.g., objective of quality, cost or quantity); such type of a context element belong to the contextual dimension of processes; (ii) *Behavioral Element*, i.e., an element related to the behavioral dimension of processes (e.g., activity execution mode or activity duration); (iii) *Resource Element*, i.e., an element related to the organizational dimension of processes (e.g., availability of a resource or experience of a human performer), and (iv) *Data Element*, i.e., an element related to the informational dimension of processes (e.g., data type or data structure).

Let's explain now the meta-relationships between versionnable concepts and context concepts. An instance of *Context Container* is associated to each instance of *Process*. Versionable components of a process are linked to one or several context elements of its corresponding context container. In addition, conditions for these context elements are to be defined. More precisely:

- The meta-class *Context Goal* allows linking *Goal Element* (a sub-meta-class of *Context Element*), on the one hand, and *Version of Process*, *Sub Process*, *Event*, *Task*, *ItemAwareElement*, *Resource* or *ResourceRole* meta-classes, on the other hand, while specifying a goal condition as a Boolean expression.
- Regarding behavioral, data and resource elements, the meta-relationships between *Context Element*, on the one hand, and *Context of Flow Node Assignment*, *Context of Data Assignment* and *Context of Resource Assignment*, on the other hand, allow specifying assignment conditions, as Boolean expressions, for linking instances of these meta-classes.

These conditions define for each process version the situation in which versions of tasks and events, versions of resources and resource roles and versions of data involved in this process have to be used.

4. NOTATION, BUILD PROCESS AND DESIGN TOOL FEATURES OF THE METHODOLOGY METHOD4BPVC

This section is dedicated to the three components of this METHOD4BPVC: notation to model versions and their contexts in accordance with BPMN4V-context meta-model; build process for building versions and deriving new versions from existing ones; design tool features.

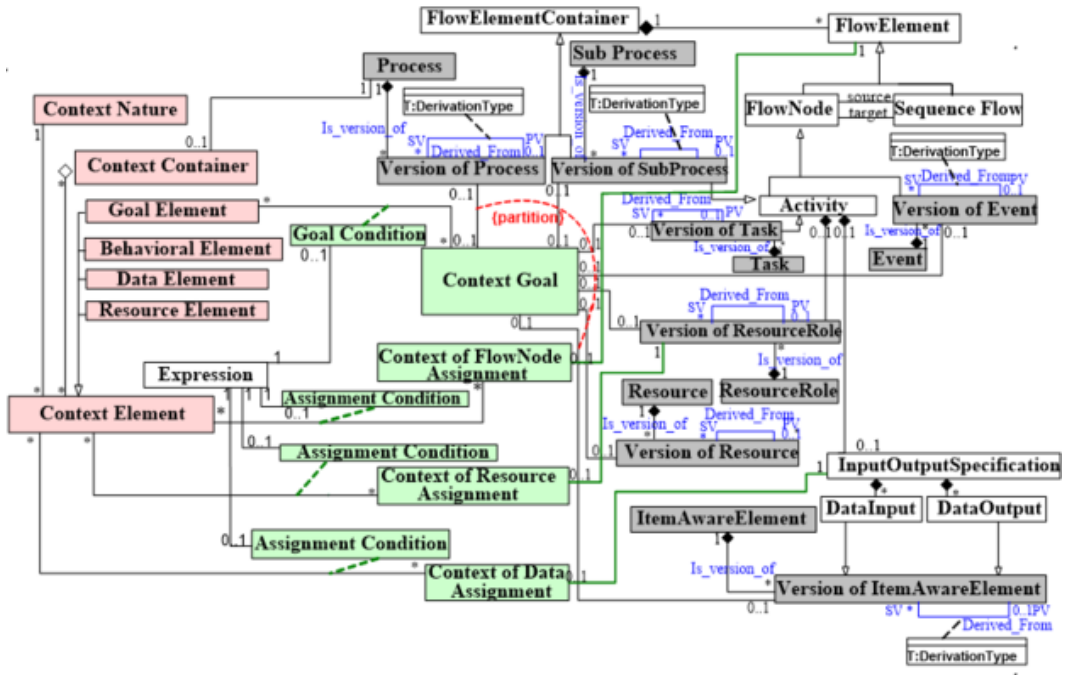
a. Modeling process version notation based on the BPMN4V-Context meta-model

It is important to provide designers with an appropriate notation for expressing versions of processes models and their contexts. The notation we propose here is named *BPMN4V-context notation*. It contains a set of concepts, a set of models and a set of use rules; concepts are used to build models while respecting use rules. This notation includes the standard BPMN2.0 notation and enriches it by the elements relating to versions and contexts. We present in the following these specific elements.

Specific concepts:

- Concept of *Version* applied to seven BPMN concepts: *Process*, *Sub Process*, *Event*, *Task*, *ItemAwareElement*, *Resource* and *ResourceRole*. Therefore, the designer can define multiple versions of constituents to better ensure the flexibility of BP.
- Concept of Version Derivation dealing with process evolution, process adaptation and process variability while keeping track of changes with a hierarchical derivation tree.

Figure 4. BPMN4V-context meta-model (Chaâbane et al., 2020)



- Concept of Derivation Type specifying if a version is an evolution or an alternative of a previous version.
- Concepts of version properties: versionId, versionCreationDate, versionCreator and versionDerivedFrom which correspond respectively to the id of a version, its creation date, the name of its creator and the version from which it is derived.
- Concept of Context that considers the necessary information, represented as context elements, used to describe the situation in which a version can be used.
- Concept of Goal that specify the purpose of the creation of a version via a set of goal elements.
- Concept of Context Assignment that makes an alliance between a version and the contextual information describing conditions of use of this version.



Specific models:

- Version of Process Model which is an enriched BPMN2.0 model composed of a set of standard BPMN constituents and new versionable constituents (i.e., Versions of tasks, Versions of sub-processes and Versions of events) linked with specific coordination pattern (Gateways and sequenceFlows).
- Context Model associated to each process. This model contains all context elements used to describe the context of this process. Thus, any context element used in the description of the context of a process must be a part of its context model.

Specific rules:

- Each versionable constituent (e.g., process, task and event) can have some versions, but each version is linked to one and only one versionable constituent.

- A version has unique id.
- Only the first concept version has a derivationSource property value equals to nil.
- Two versions of the same versionable constituent must have different context and as consequence different definitions. For instance, the context of two versions of the same task must be defined using different context elements values. This implies that these two versions must be different either in their type or in the involved versions of ResourceRoles, or in the consumed/produced ItemAwareElements.
- A version of task is an activity that may have a context goal within a process version or a sub-process version according to a goal element
- A version of ResourceRole can be assigned to a particular version of task only if assignment conditions (defined using context elements) are checked.

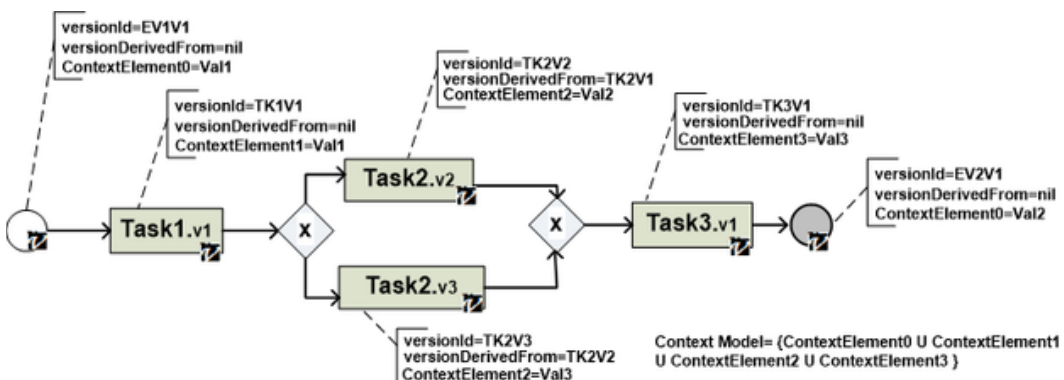
Graphically, we decorate versionable concepts with this icon . For instance, version of start event will be presented by the following sheep  in the BPMN4V-context notation. The version icon is not applicable for a non versionable concept such as Gateways or sequenceFlows. Moreover, we add versions properties (versionID, versionCreationDate, derivationSource, versionType) of each versionable concepts. We present in Figure 5 an example of an abstract version of process modeled according to *BPMN4V-context notation*. In this example we present versionable constituents (tasks and events) along with context elements describe context of these constituents. As for context model, it contains all context elements used in the definition of the context of the presented version of process.

b. Build process

To define a new version in a multi-version environment where several versions co-exist, it is important to ask the following questions: “Why a version is derived from a specific previous one? This new version is an alternative or an evolution?”. Therefore, when there is a need of derivation, it is crucial to help BPM designers to retrieve the most appropriate version to be the source of derivation and the type of this derivation in order to have a comprehensive derivation hierarchy of versions that mark their evolutions. This should only be envisaged if each version is linked to its context.

To reply to these questions, we propose the build process, given in Figure 6, that include five steps. The main objective sought for this process is to help BPM designers in the derivation of versions since we believe that a version could not be properly derived from whatever other one.

Figure 5. Example of an abstract version of process model modeled using BPMN4V-context notation



In the first step of the build process, the BPM designer proceeds to the modelling of the first versions of BP processes according to BPMN4V-context notation. The creation of the other versions is done by derivation using the flowing steps.

In the second step, specifies a query based on the contextual elements describing the current situation. This query is used for querying the Versions-Database to retrieve the most appropriate versions that can be suitable for the specified context. We note that Versions-Database, which implements the BPMN4V-context meta-model, contains the versions previously modeled as instances of this meta-model. To retrieve versions, we propose a set of algorithms allowing comparisons between the user context (specified in the query) and the contexts of versions stored in Versions-Database. The result of this step is a set of versions sorted in descending order (from the version that has the closest context).

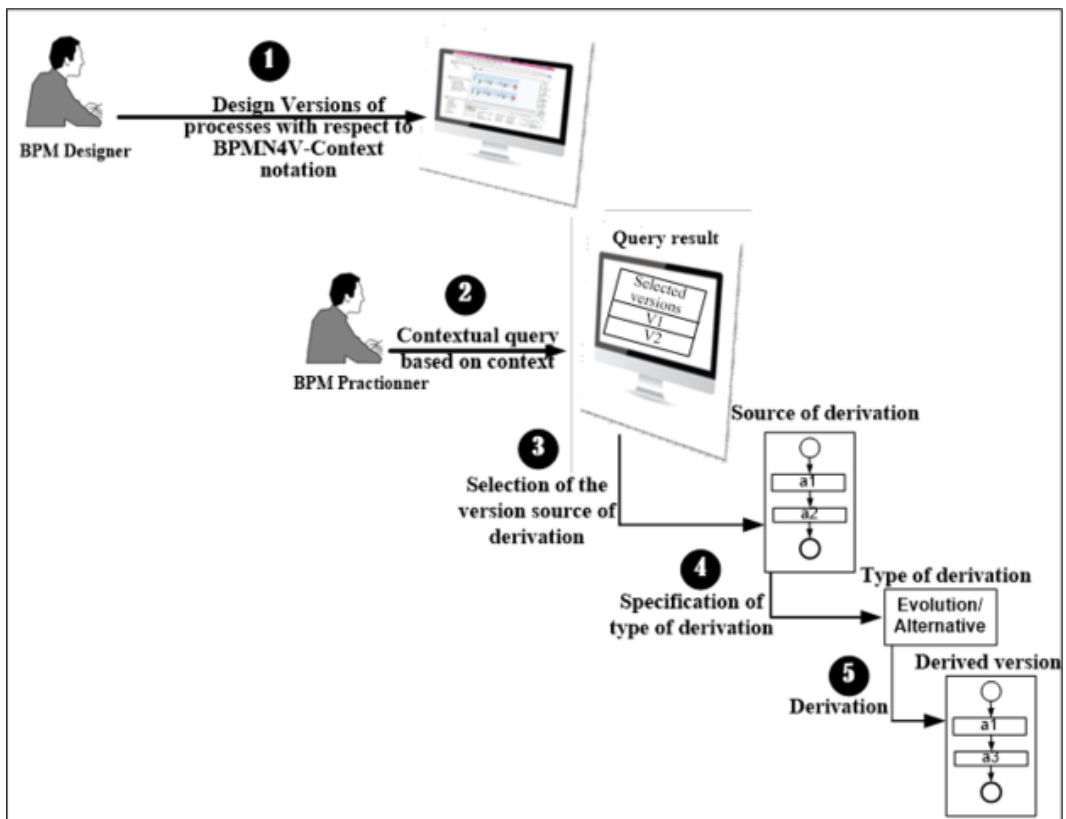
In the third step, the BPM designer selects from the retrieved versions the most appropriate one to be the derivation source.

The fourth step allows notifying the derivation type: alternative derivation or evolution derivation. It is based on the analysis of the context elements expressed in the definition of the current situation given by the designer in step 1.

The fifth step consists on creating a new version by derivation from the version specified in step 3 and using the derivation type identified in step 4.

1. Derivation source retrieving

Figure 6. Build process



This step aims to select candidate versions that can be the source of derivation for a specific new version. First, BPM designer defines a query containing a set of context elements that describe the context of the current situation requiring derivation. Candidate versions for derivation can be identified by comparing the query context with contexts of existing versions (i.e., previously modeled versions stored in BPMN4V-context DataBase). To retrieve candidate versions for derivation, we propose two algorithms: selectDerivationSource (Algorithm 1) and contextCompare (Algorithm 2).

Algorithm 1

```
selectDerivationSource (userContext: Context, p: Process):  
Version_of_Process[]  
Local score:Integer  
    candidateVersions:Version_of_Process[]  
Begin  
List<Version_of_Process> versionsList= p.getVersions()  
    If versionsList.size()==1 //case of the first version  
derivation  
        candidateVersions.setItem(versionsList.  
getItem())  
    else  
        For each v in versionsList //context compare  
to find source  
            score=  
ContextCompare(userContext,v.getContext())  
            sort(candidateVersions, score,v)  
        End for  
    Endif  
return candidateVersions  
End
```

The parameters of selectDerivationSource algorithm are: (i) userContext which corresponds to a set of elements that describe the context specified by the designer, and (ii) the process p that needs derivation of one of its versions. First, it stores in versionsList all versions of p using the getVersions() function.

If versionList contains just one version, this later has to be considered as the source of derivation since there are no comparisons to do. But if versionList contains two or more versions, selectDerivationSource algorithm compares the context of each of these versions of the process p (given by the getContext() function) with the userContext using the contextCompare() function. This later returns a score marking the degree of similarity between these two contexts. Then, the selectDerivationSource algorithm call the sort function for an orderly insertion in the candidateVersions list of this version among the versions already selected of the process p according to their scores. Finally, selectDerivationSource returns candidateVersions list sorted form the version with the highest score to version with the lowest one.

Regarding, the contextCompare algorithm, given below, it allows comparing two different contexts, each one is specified with a set of context elements (parameters). This algorithm compares the parameters names and values, and calculates a comparison score: 1 point is added to this score for each context element with the same name and the same value; 0.5 is added to the score for each context element with the same name but different values; otherwise, the score is not incremented. To do so, contextCompare algorithm uses the following functions:

- getParameters(): returns the set of parameters of a particular context.
- getParamName(): returns the name of a specific parameter.
- getParamValue(): returns the value of a specific parameter.

- Verif(prv1, prv2): returns true if the parameter value prv1 satisfies the condition related to the parameter value prv2, else it returns false.

Algorithm 2

```
ContextCompare(Context c1, Context c2): Integer
Local score: Integer
Begin
score=0
List<Parameter> lp1=c1.getParameters()
List<Parameter> lp2=c2.getParameters()
For each p1 in lp1
    For each p2 in lp2
        //compare context elements names and values and calculate
score
        If p1.getParamName()==p2.getParamName()
            If verif(p1.getParamValue(),p2.getParamValue())
                score=score+1
            Exit for
        else
            score=score+0.5
            Exit for
        End if
    End if
End for
return score
End
```

2. Specifying the type of derivation

As indicated before, in the second step of the proposed derivation process, BPM designer selects from the returned candidate versions (of step 1) one version to be the source of the derivation. We denote vs this version. The third step of the recommended build process consists on identifying the type of derivation (evolution or alternative). In fact, when a new version vd has to be derived from vs, we have to consider the following question: “is vd an evolution of vs or an alternative?”. In order to reply to that question, we propose the selectDerivationType algorithm given below.

According to this algorithm, the type of derivation depends on (i) the contextual information that triggers the change from one version (vs) to another (vd), and (ii) the type of this change (temporary/permanent).

If the contextual information that triggers the need of derivation comes from immediate context or environment context, the type of derivation is alternative. This is because changes coming from immediate context (e.g. resource unavailability) or environment context (e.g., inundation) lead to occasional situations that directly impact the process definition on short-term. Generally, these changes require the creation of alternative versions used in infrequently situations.

However, if the contextual information that triggers the change from one version to another comes from Internal context or External context, we have to verify the type of change (temporary or permanent). In the case of, the change is temporary the type of derivation is alternative, otherwise when the change is permanent the derivation type is evolution. In fact, contextual information coming from Internal context (e.g., business goals) or External context (e.g., customer requirements) corresponds to strategic information that require decision-making at short-time or long-time. Thus,

we recommend the creation of an alternative version when there are temporary changes causing the derivation need and an evolution version if these changes are permanent.

Algorithm 3

```
String selectDerivationType (Version vs, Version vd,
ContextElement trigger, String changeType)
Begin
List Imc = getAllImmediateContextElements()
List Inc = getAllInternalContextElements()
List Exc = getAllExternalContextElements()
List Enc = getAllExternalContextElements()
If trigger  $\square$  Imc or trigger  $\square$  Enc then
    return "alternative" // vd is an alternative of vs
Else
if trigger  $\square$  Inc or trigger  $\square$  Exc then
    if changeType ="temporary" then
        return "alternative" // vd is an alternative of vs
    else
        return "evolution" // vd is an evolution of
vs
        End if
    End if
End if
End
```

3. Derivation

Derivation is the final step in the proposed build process which aim to create a derived version based on the results given by previous steps of this process mainly the version source of derivation and the derivation type. To better illustrate this step, we present below *Derive* algorithm. In this later we start by making a copy of the version source of derivation *vs* using clone() function. The cloned version, stored in *vd* variable, has initially the same definition as *vs*. Then, the algorithm proceeds to the update of the derived version according to its type. For instance, if the derived version is a version of process, the algorithm allows (i) the update of its activities and/or its events and (ii) the update of its coordination pattern used to link these activities and events. Algorithms of Update operations on versions are detailed in (Weber et al., 2008)[33]. The update of *vd* has to be repeated until there is difference between the definitions of *vd* and *vs*. This can be done using checkDifference(version1, version2) function which return true if two versions have the same definition and false otherwise. Finally, *vd* has to be related to the derivation hierarchy of *vs* using the function bind(derivedVersion, derivationType). Under this function *vd* will be linked to *vs* using the derivationType that can be "evolution" or "alternative".

Algorithm 4

```
Version Derive (Version vs, String Type)
Local Version vd
Begin
    vd=vs.clone()
    do
        Begin
            Swith (vd)
                case: vd isInstanceOf Version_of_Process
                    Begin
```

```

        vd.updateActivities()
        vd.updateEvents()
        vd.updatePatterns()
    End
case: vd isInstanceOf Version_of_Task
    Begin
        vd.updateResourceRoles()
        vd.updateItems()
    End
case: vd isInstanceOf Version_of_Event
    Begin
        vd.updateType()
        vd.updateInformation()
End
    case: vd isInstanceOf Version_of_Resource
        vd.updateParameters()
    case: vd isInstanceOf Version_of_ResourceRole
        Begin
            vd.updateType()
            vd.updateResource()
        End
    case: vd isInstanceOf Version_of_ItemAwareElement
        Begin
            vd.updateType()
            vd.updateStructure()
        End
    End
    While (!(ckeckDifference(vs, vd)))
    vs.bind(vd, type)
    Return vd
End

```

c. Design tool features

Regarding the third component of the METHOD4BPVC, it corresponds to a software tool ensuring two main functions. It:

- 1- provides a modeler that helps BP designers to define version process models and context models according to the BPMN4V-Context Notation and respects the defined rules,
- 2- automates the proposed build process through the implementation of its algorithms.

5. PROOF OF CONCEPT: THE KITCHEN-MAKING CASE STUDY

To better illustrate the proposed METHOD4BPVC, and specially its build process, we introduce in this section the *kitchen-making* (KM) case study. This case study supports the construction of kitchens and takes place within “IPEA Cuisine”, a company providing specialized and integrated services in the interior design filed. Initially, IPEA Cuisine manufactures and installs standard and personalized kitchens. However, it does not provide painting service. For this reason, it has a subcontract with “Uni Paint”, a company that provides professional printing services, to paint the manufactured kitchens.

More precisely, the first version of KM process (KM.v1), presented in Figure 7, consists of six activities: *Kitchen design*, *Machining and manufacturing*, *Truck loading*, *Assembly of accessories*, *Kitchen installation* and *Quality checking*. First, a *IPEA Cuisine* designer prepares a kitchen plan describing dimensions and colors. Then, it proceeds to the manufacturing of plates that make up the kitchen. As *IPEA Cuisine* was not able to provide painting service, the adopted strategy was to subcontract painting plates. For this reason, *IPEA Cuisine* has to transport plates for painting using its own trucks to the *Uni Paint* company. After receiving painted plates, it proceeds to accessories assembly. In the next activity, workers of *IPEA Cuisine* install the kitchen in the client house. Finally, a design engineer from *IPEA Cuisine* controls kitchen making and Installation. The event that deficiencies are identified in kitchen manufacturing and installation is repeated. Context of the first version of KM process is defined using the following context elements:

- IPEA Cuisine strategy (from Internal context) = Painting subcontracting
- IPEA Cuisine Vehicle availability (from Immediate context) = available
- Engineer availability (from Immediate context) = available

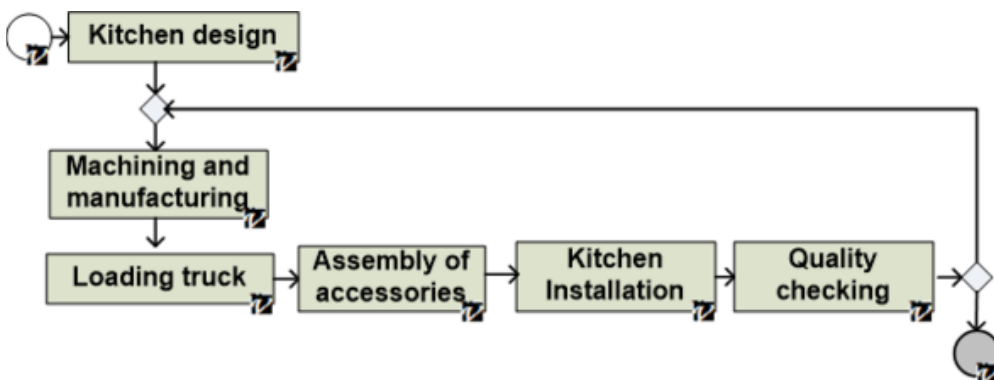
The second version of KM process (KM.v2) is used when *IPEA Cuisine* trucks are not available. For not wasting time, *Uni Paint* company can ensure plates transport if one of its vehicles is available. Thus, the context of the second version of KM process is defined as follow:

- IPEA Cuisine strategy (from Internal context) = Painting subcontracting
- IPEA Cuisine Vehicle availability (from Immediate context) = Not available
- Uni Paint Vehicle availability (from Immediate context) = available
- Engineer availability (from Immediate context) = available

According to the selectDerivationSource algorithm, KM.v2 is derived from KM.v1, since it represents the second version that has to be derived from the first one. Likewise in accordance with this algorithm, this version is alternative since the change is triggered by “Vehicle availability” context element which is an element from immediate context.

When both *IPEA Cuisine* and *Uni Paint* trucks are not available, IPEA Cuisine contacts a trucking company to transport plates to Uni Paint company. Thus, a third version of KM process (KM.v3) has to be defined by derivation from KM.v1 or KM.v2. To do so, we have to use selectDerivationSource algorithm to identify the most suitable version that can be the source of derivation of KM.v3. This algorithm compares context of KM.v3 with contexts of KM.v1 and

Figure 7. The first version of KM process model



KM.v2 and calculates scores that represent similarity degrees between these contexts. Let us remind that an elementary score takes 1 point if the considered versions have the same context element with the same value; it takes 0.5 point when these versions have the same context element but with different values; otherwise, it takes 0 point.

Table 2, presented below, gives the context of KM.v3 and explains how calculate scores of KM.v1 and KM.v2. For example, the context element “IPEA Cuisine Vehicle availability” takes 0.5 point for KM.v1 since it is used in both versions (i.e., KM.v1 and KM.v3) with different values. Whereas this element takes 1 point for KM.v2 as it has the same value in KM.v2 and KM.v3.

According to Table 2, KM.v3 has to be derived from KM.v2 as it has the highest score. According to the selectDerivationType algorithm, this version is alternative since the change is also triggered by “Vehicle availability” context element which is an element from immediate context. Figure 8 shows the definition of the third version of KM process.

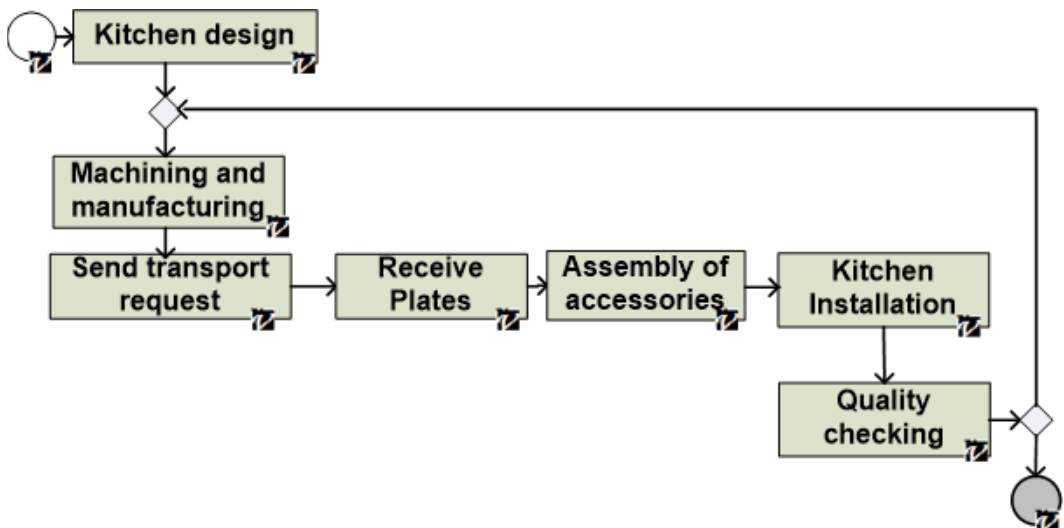
Regarding the kitchen control, *IPEA Cuisine* requires that the design engineer performs quality checking as indicated in KM.v1 version. But since this engineer should receive continuing further training, he becomes temporary unavailable. In order to avoid delay kitchen delivery, an *IPEA Cuisine* worker that has more than 10 years’ experience can replace the engineer and perform the control activity. Thus, a new version (KM.v4) has to be defined.

The context of KM.v4 version can be defined using the following context elements:

Table 2. Result of the comparison of the context of KM.v3 against those of KM.v1 and KM.v2

KM.v3 context	KM.v1	KM.v2
IPEA Cuisine strategy= Painting subcontracting	1	1
IPEA Cuisine Vehicle availability = Not available	0.5	1
Uni Paint Vehicle availability = Not available	0	0.5
Engineer availability= available	1	1
Total Score	2.5	3.5

Figure 8. The third version of KM process model



- IPEA Cuisine strategy (from Internal context) = Painting subcontracting
- IPEA Cuisine Vehicle availability (from Immediate context) = available
- Design Engineer availability (from Immediate context) = Not available
- IPEA Cuisine Worker = available
- Worker experience > 10 years

To be defined, KM.v4 has to be derived from a previous version of KM process (i.e., from KM.v1, KM.v2 or KM.v3). Table3, presented below, gives results given by the selectDerivationSource algorithm corresponding to a comparison of the context of KM.v4 version with those of KM.v1, KM.v2 and KM.v3 versions. In accordance with this table, KM.v4 has to be derived from KM.v1, and it is alternative since the change is triggered by “Engineer availability” context element which is an element from immediate context.

With the emergence of the use of melamine in kitchen manufacturing, *IPEA Cuisine* changes its strategy of work and decides to adopt this technique in its factories. Since kitchens manufactured with melamine do not require to be painted, *IPEA Cuisine* no longer subcontracts painting. Therefore, a new version of KM process model (KM.v5) is derived. Context of KM.v5 version is defined as follow:

- IPEA Cuisine strategy (from Internal context) = Produce kitchen with melamine
- Design Engineer availability (from Immediate context) = available

Table 4, given below, summarizes the scores given by selectDerivationSource algorithm. According to this table, the designer can choose from KM.v1, KM.v2 or KM.v3 to derive KM.v5. Also, this version is an evolution one since the change is permanent and it is triggered by “Strategy” context element which is an element from internal context.

Referring to build process steps, the derivation hierarchy of all versions of KM process model, shown in Figure 9, illustrates relationships between versions as well as their derivation type.

6. BPMN4V-CONTEXT ARCHITECT

BPMN4V-context Architect is a visual modeling and design tool based on the BPMN4V-context notation. The platform, implemented using MVC pattern, supports modeling and handling versions of business process models with respect to the BPMN4V-context meta-model and the build process. Figure 10 shows the architecture of this tool.

The view part of BPMN4V-context Architect offers two kinds of interfaces: (i) BPMN4V-context Forms, that gives a set of Graphical User Interfaces (GUI) allowing the retrieve and derivation of versions, and (ii) BPMN4V-context Modeler, which is an editor for creating and handling versions in accordance with BPMN4V-context notation.

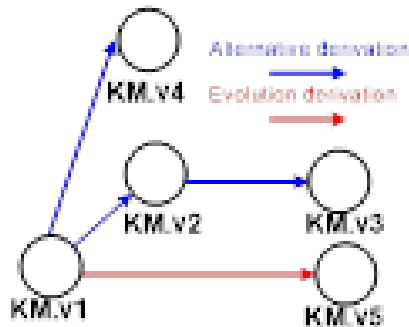
Table 3. Result of the comparison of the context of KM.v4 against those of KM.v1, KM.v2 and KM.v3

	KM.v1	KM.v2	KM.v3
KM.v4	2.5	2	2

Table 4. Result of the comparison of the context of KM.v5 against those of those of KM.v1, KM.v2, KM.v3 and KM.v4

	KM.v1	KM.v2	KM.v3	KM.v4
KM.v5	1.5	1.5	1.5	1

Figure 9. Derivation hierarchy of versions of the KM process Model



The model part of BPMN4V-context Architect manages the BPMN4V-context database, which implements the BPMN4V-context meta-model, that stores all versions modeled using the view part. It is an XML database that contains the descriptions of previously modeled versions of process models as well as their contexts.

The controller component of BPMN4V-context Architect has the following functions: (i) it checks whether versions of process models are or not well modeled with respect to the BPMN4V-context notation, (ii) it ensures the insertion, update and deletion of versions in/from the BPMN4V-context database and (iii) it ensures the build process by applying the selectDerivationSource algorithm.

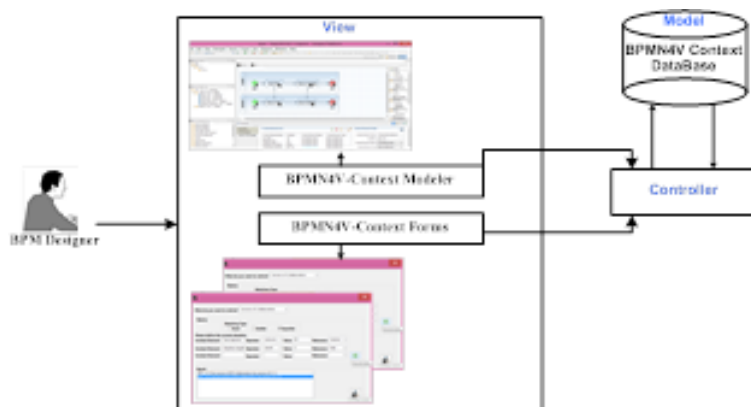
We detail in the next sub sections how BPMN designers can take advantage of BPMN4V-context Modeler and Forms to create and derive versions of process models.

a. BPMN4V-context Modeler Overview

BPMN4V-context Modeler is implemented as an extension of the already existing Eclipse BPMN Modeler plug-in. It extends this later by integrating new widgets showing versions details. Figure 11 below is a screenshot visualizing the first version of the KM process model created with this editor.

The central part of the screenshot (part E) is the drawing canvas which provides multiple tabs, each one being used to model and display a separate version of a process model. To highlight the notion of version, we decorate shapes of versionable concepts (i.e., process tab name, activity shape

Figure 10. Architecture of BPMN4V-Context Architect



and the event shape) with the following icon . This icon is used to mark that any element shown in the drawing canvas is a version. The right part of Figure 11 (part) represents the *Tool Palette*, which contains tools that can be dragged onto the drawing canvas to add BPMN4V-context concepts.

The bottom part (part Z) of the Figure 11 gives an Eclipse property tab, named “*Context Information*”, that defines the context of the first version of the KM process model. This property tab concerns versions of processes as well as other versionable concepts. It shows the list of context elements and conditions that define the context of the activated version in the drawing canvas. Particularly, this list contains the name, value, nature (i.e., immediate, internal, external or environmental context) and type (i.e., Goal, Behavioral, Data or Resource) of each context element used in the definition of the context of the selected version. Buttons above allow adding, deleting, moving or editing context elements. In the right part of this property tab, the user can either introduce a new context element and condition by defining its name, value, type, and nature, or edit an existing one. Once the version of a process model is designed and its context is well defined in the *Context Information* tab, the user can save it using the save button of the editor; as result, the version is inserted in the BPMN4V-context database.

The left parts of the Figure 11 (parts , and ‘) correspond to new Eclipse views showing versions details. More precisely, we have defined three Eclipse views which are *Version Data View*, *List of Hierarchy View* and *Activities View*.

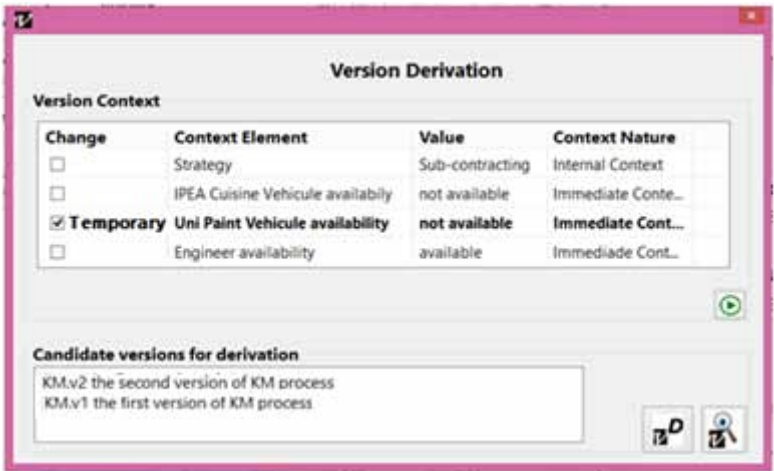
The *Versions Data View* indicates properties (name, version id, version state and version details) of the active version of the process model shown in the drawing canvas.

For example, the *Versions Data View* presented in Figure 11 part indicates that the active version, identified by KM.v1, corresponds to the first version of the KM process model. In addition, this view details each sub-process, task and event that make up the considered version.

The *Hierarchy View* provides a hierarchical tree-oriented view representing the derivation hierarchy of the active versionable concept (e.g., *Process* or *Task*) of the drawing canvas. In Figure 11 part , the *Hierarchy View* shows the derivation hierarchy of the KM process since the active versionable concept is this process.

The *List of Activities View*, presented in Figure 11 part ‘, displays all the previously modeled tasks and their corresponding versions. This view allows designers to reuse previously modeled versions by dragging and dropping them into the drawing canvas.

Figure 11. BPMN4V-context Modeler overview



Finally, BPMN4V-context Modeler provides the “*Handle versions*” contextual menu to perform derivation of versions as shown in Figure 11 part '. This menu holds for each versionable concept of the BPMN4V-context notation and implements derivation algorithms.

b. Modeling versions of the KM process model

The first version of the KM process model is created from scratch using the palette (for adding new component) or the *List of Activities View* (for integrating existing versions). The drawing canvas of Figure 11 contains the result of the creation of KM.v1.

The second version of KM process model is directly derived from the first one since there is no comparison to do. This derivation is performed using the derive command of the *Handle versions* context menu and it results in the creation of a new tab that contains a copy of the derived version. More precisely, when the designer derives KM.v1, a new tab named KM.v2 appears in the drawing canvas, initially having the same definition of KM.v1. The designer should then make changes to KM.v2 by adding, modifying and/or removing some components (such as tasks, events, etc.) according to the specification of the second version of the KM process model described in section 5.

The other versions of the KM process model are created by derivation from previous versions. So, it is crucial to retrieve the most suitable version that has to be the source of derivation. For instance, for modeling the third version KM.v3, it is necessary to decide if this version has to be derived from KM.v1 or from KM.v2, and if it is an alternative version or an evolution one. To do so, we propose Drive GUI, presented in Figure 12, for deriving versions. This GUI implements the derivation process and helps the designer to (i) choose the appropriate version as source of derivation, (ii) select a particular version, (iii) identify the type of this derivation (alternative or evolution) and (iv) perform the derivation.

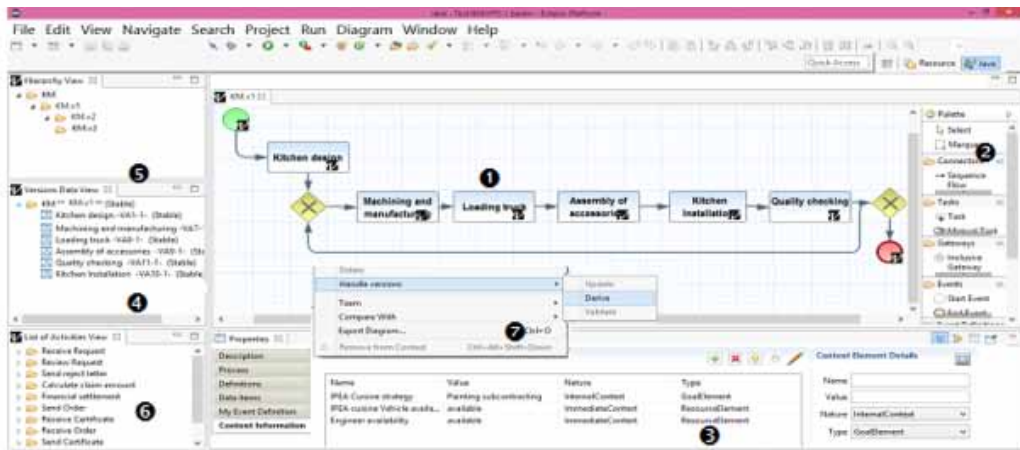
When a designer uses Derive GUI to derive a version of a process model, it has to define the situation that leads to the derivation of this new version by specifying the concerned context elements and their values. In this definition the designer has to highlight the more pertinent context element that triggers the derivation need. This later is used by the selectDerivationType algorithm in order to decide the derivation type. Then, the designer should click on the *Execute* button to trigger the selectDerivationSource algorithm and display the list of “*Candidate versions for derivation*” containing the retrieved versions. The user can select a version from this list and click on the *Visualize* button to show its schema using BPMN4V-context modeler. Finally, the user clicks on *Derive* button to derive the selected version.

For instance, in Figure 12 the current situation presented corresponds to the case when both *IPEA Cuisine* and *Uni Paint* trucks are not available. This situation leads to the creation of the third version KM.v3, which is an alternative version derived from KM.v2, as shown in the *Candidate versions for derivation* list of Figure 12. When the user select KM.v2 and click on *Derive* button, a new tab named KM.v3 appears in the drawing canvas of BPMN4V-context Modeler. KM.v3 has the same definition as KM.v2 until changed to be validated.

7. CONCLUSION

Business Processes flexibility continue to be a major challenge since it is crucial to consider the current environment, which is becoming very fluctuating, and to adapt processes accordingly. Faced with this challenge, several literature contributions have recommended versioning (Chaâbane et al., 2010; Ekanayake et al., 2011; Ellouze et al., 2016; Lassoued et al., 2016; Zhao & Liu, 2013) as a key notion to deal with process flexibility issue. However, to our knowledge, none of them offered a design methodology to help designers define and handle model versions for processes. Thus, the contribution we have proposed in this paper to solve the process adaptation issue is twofold: a design methodology combined with a derivation approach to properly address process flexibility at design-time.

Figure 12. GUI for BP derivation



The proposed design methodology, entitled METHOD4BPVC, has three components: (i) a notation to model process versions, (ii) a build process for deducing versions of process models and (iii) a tool implementing the proposed notation and build process.

The richness and effectiveness of this notation come from the fact that it is based on the proposed meta-model in (Chaâbane et al., 2020), named BPMN4V-context, which is an extension of BPMN2.0 meta-model enriching its concepts, models and construction rules to be able to define and manage versions of process models and their contexts. This solution benefits from two factors. First, it is based on BPMN2.0 which is the de facto standard for modeling processes, and therefore can be easily applied for multiple domains and applications. Second it overcomes the weakness of BPMN2.0 by supporting the context awareness, which is an important dimension to be considered in process modeling since it represents a bridge between processes and their environments, and thus ensures their efficiency and their applicability.

As for the derivation approach, it has allowed us to provide a context-based build process that helps designers to define, derive, and implement model versions in order to ensure the flexibility of processes. Indeed, the versioning makes it possible to (i) follow the evolutions of process models by defining versions for each significant change, (ii) reuse, if necessary, these versions to suit the current context, and (iii) consider process evolution, process adaptation and process variability, which are the main process flexibility needs as proposed in (Reichert & Weber, 2012).

Finally, BPMN4V-context Architect is a specific tool for modeling and deriving versions of process models. It helps designers to define solutions ensuring the context awareness and thus the process flexibility.

However, the contribution of this paper has limitations that we plan to overcome in our future work as follows. On the one hand, the paper focuses only on private processes, i.e., business processes that hold in one organization. But it is also necessary to examine flexibility of collaborative processes, i.e., processes that involve several organizations. On the other hand, the proposed solution focuses only on flexibility at design-time. Nevertheless, it is crucial to deal with flexibility at run-time to support changes that occur on running processes. This can be achieved by defining an adaptation engine that takes advantages of the versions of each considered process model, and that interoperates with multiple execution engines (Song et al., 2019).

REFERENCES

- André, P., & Vailly, A. (2001). *Conception des systèmes d'information-Panorama des méthodes et des techniques*. Eyrolles.
- Aysolmaz, B., Schunselaar, D.M., Reijers, H.A., & Yaldiz, A. (2017). Selecting a process variant modeling approach: guidelines and application. *International Journal of Software & Systems Modeling*, 18(2), 1155-1178.
- Barba, I., Jiménez-Ramírez, A., Reichert, M., Del Valle, C., & Weber, B. (2021). Flexible runtime support of business processes under rolling planning horizons. *Expert Systems with Applications*, 177(11), 114857. doi:10.1016/j.eswa.2021.114857
- Ben Said, I., Chaâbane, M. A., Andonoff, E., & Bouaziz, R. (2018). BPMN4VC-modeller: Easy-handling of versions of collaborative processes using adaptation patterns. *International Journal of Information Systems and Change Management*, 10(2), 140–189. doi:10.1504/IJISCM.2018.094604
- Ben Said, I., Chaâbane, M. A., Bouaziz, R., & Andonoff, E. (2016). A Version-based Approach to Address Flexibility of BPMN Collaborations and Choreographies. *Proceedings of the 13th International Joint Conference on e-Business and Telecommunications*, 31-42.
- Chaâbane, M.A., Andonoff, E., Bouaziz, R., & Bouzguenda L. (2010). Modélisation multidimensionnelle des versions de processus. *Journal on Ingénierie des Systèmes d'Information*, 15(5), 89–114.
- Chaâbane, M. A., Ben Said, I., Ellouze, F., Bouaziz, R., & Andonoff, E. (2020). A context-based approach for modelling and querying versions of BPMN processes. *International Journal of Business Process Integration and Management*, 10(1), 62-86.
- Ekanayake, C., La Rosa, M., Ter Hofstede, A. H., & Fauvet, M. C. (2011). Fragment-based version management for repositories of business process models. *International Conferences OTM Confederated On the Move to Meaningful Internet Systems*, 20–37. doi:10.1007/978-3-642-25109-2_3
- Ellouze, F., Chaâbane, M.A., Andonoff, E., & Bouaziz, R. (2017). Onto-vP2M: a new approach to model and manage collaborative process versions using contexts and ontologies. *International Journal of e-Collaboration*, 13(3), 39–62.
- Ellouze, F., Chaabane, M. A., Bouaziz, R., & Andonoff, E. (2016). Addressing inter-organisational process flexibility using versions: the VP2M approach. *Proceedings of the International Conference on Research Challenges in Information Science*, 1–12. doi:10.1109/RCIS.2016.7549280
- Hallerbach, A., Bauer, T., & Reichert, M. (2008). Context-based Configuration of Process Variants. *ICEIS Workshop on Technologies for Context-Aware Business Process Management*, 31-40.
- Hallerbach, A., Bauer, T., & Reichert, M. (2010). Configuration and Management of Process Variants. *Handbook on Business Process Management*, 237-255.
- Jemal, L., Saidani, O., & Nurcan, S. (2018). Flexibility in Business Process Modeling to Deal with Context-Awareness in Business Process Reengineering Projects. *Proceedings of the 19th International Conference International Conference on Business Process Modeling, Development and Support BPMDS*, 35-48.
- Lassoued, Y., Bouzguenda, L., & Mahmoud, T. (2016). Context-aware business process versions management. *International Journal of e-Collaboration*, 12(3), 7–33.
- Liaskos, S., Lapouchnian, A., Yu, Y., Yu, E., & Mylopoulos, J. (2006). On Goal-based Variability Acquisition and Analysis. *Proceedings of the International Conference on Requirements Engineering*, 76–85.
- Milani, F.P., Dumas, M., Ahmed, N., & Matulevičius, R. (2016). Modelling families of business process variants: A decomposition driven method. *International Journal of Information Systems*, 56, 55-72.
- Natschläger, C., Geist, V., Illibauer, C., & Hutter, R. (2016). Modelling business process variants using graph transformation rules. *Proceedings of the 4th international Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, 65-74.
- Nurcan, S., & Edme, M. H. (2005). Intention Driven Modelling for Flexible Workflow Applications. *International Journal on Software Process, Improvement, and Practice*, 10(4), 363–377. doi:10.1002/spip.240

- OMG. (2014). *Business Process Model and Notation (BPMN 2.0)*. <http://www.omg.org/spec/BPMN/2.0>
- Reichert, M., Hallerbach, A., & Bauer, T. (2015). Lifecycle Management of Business Process Variants. Handbook on Business Process Management. doi:10.1007/978-3-642-45100-3_11
- Reichert, M., & Weber, B. (2012). *Enabling Flexibility in Process-Aware Information Systems: Challenges, Methods, Technologies*. Springer. doi:10.1007/978-3-642-30409-5
- Rosa, M. L., Dumas, M., & Hofstede, A. H. (2009). Modeling Business Process Variability for Design-Time Configuration. Handbook of Research on Business Process Modeling, 204-228. doi:10.4018/978-1-60566-288-6.ch009
- Rosemann, M., & Recker, J. (2006). Context-aware Process Design Exploring the Extrinsic Drivers for Process Flexibility. *Proceedings of the international Conference on Business Process Modeling, Development and Support CAiSE*, 149-158.
- Rosemann, M., Recker, J., & Flender, C. (2008). Contextualisation of business processes. *International Journal of Business Process Integration and Management*, 3(1), 47-60.
- Saidani, O., & Nurcan, S. (2009). Context-Awareness for Adequate Business Process Modelling. *Proceedings of the International Conference on Research Challenges in Information Science*, 177-186.
- Saidani, O., Rolland, C., & Nurcan, S. (2015). Towards a generic context model for BPM. *Proceedings of Hawaii Conference on System Sciences*, 4120-4129. doi:10.1109/HICSS.2015.494
- Santos, E., Pimentel, J., Castro, J., Sánchez, J., & Pastor, O. (2010) Configuring the Variability of Business Process Models Using Non-Functional Requirements. *Proceedings of the international Conference on Business Process Modeling, Development and Support CAiSE*, 274-286.
- Song J., Weifang Z., & Yiran J. (2019). Design of Workflow Engine Based on Relational Structures. *International Journal of Advanced Pervasive and Ubiquitous Computing*, 11(4), 34-43.
- Tealeb, A., Awad, A., & Galal-Edeen, G. H. (2014). Context-Based Variant Generation of Business Process Models. *Proceedings of the international Conference on Enterprise, Business-Process and Information Systems Modeling BMMDS/EMMSAD*, 363-377.
- van Eck, M., Sidorova, N., & van der Aalst, W. (2015). KPI-based Activity Planning for People Working in Flexible Processes. *Proceedings International Conference on Advanced Information Systems Engineering co-located*, 97-104.
- Viknesh A., & Yoke Kin W. (2020). Distinguishing approach, methodology, method, procedure and technique in process systems engineering. *International Journal on Clean Technologies and Environmental Policy*, 22(3), 547-555.
- Weber, B., Reichert, M., & Rinderle-Ma, S. (2008). Change patterns and change support features - Enhancing flexibility in process-aware information systems. *International Journal of Data & Knowledge Engineering Data*, 66(3), 438-466.
- Zhao, X., & Liu, C. (2013). Version management for business process schema evolution. *International Journal of Information Systems*, 38(8), 1046-1069.

Imen Ben Said received her PhD in Computer Science from the University of Sfax and University of Toulouse 1 Capitole in 2017. Since 2009, she was a teacher/researcher at the Faculty of Economics and Management, University of Sfax. She is a Member of the MIRACL Laboratory. Her research concerns are information systems modelling and business process management (BPM). Currently, her works are directed towards the flexibility of business processes using the version notion.

Mohamed Amine Chaâbane received a PhD degree from University of Toulouse, France in 2012. He is an associate professor at Higher Institute of Business Administration, University of Sfax – Tunisia and member of MIR@CL (Multimedia, InfoRmation systems and Advanced Computing Laboratory). His research interests focus on Business Process Management field. He is working on topics related to Business Process Modeling, Process Flexibility, Context of Process and Self-Adaptation of Process.

Rafik Bouaziz is full professor on computer science at the Faculty of Economic Sciences and Management of Sfax University, Tunisia. He was the president of this University during August 2014 – December 2017, and the director of its doctoral school of economy, management and computer science during December 2011 – July 2014. His PhD has dealt with temporal data management and historical record of data in Information Systems. The subject of his accreditation to supervise research was “A contribution for the control of versioning of data and schema in advanced information systems”. Currently, his main research topics of interest are temporal databases, real-time databases, information systems engineering, ontologies, data warehousing and workflows. Between 1979 and 1986, he was a consulting Engineer in the organization and computer science and a head of the department of computer science at CEGOS-Tunisia.