EasyChair Preprint

№ 4381

A Hybrid Approach based on Reuse Techniques for Autonomic Adaptation of Business Processes

Jamila Oukharijane, Mohamed Amine Chaabâne,
Imen Ben Said, Eric Andonoff and Rafik Bouaziz

October 12, 2020

A Hybrid Approach based on reuse techniques for Autonomic Adaptation of Business Processes

Abstract. Complexity of highly dynamic environments in which processes operate makes their adaptation difficult to perform. However, adaptation of an in-progress process is an essential requirement for any Business Process Management System (BPMS). So, several contributions have recommended the use of the MAPE-K (Monitor, Analyze, Plan, Execute - Knowledge) control loop from the autonomic computing as a solution to tackle this issue, thus bringing BPMS with self-adaptation capabilities. However, in these contributions, a comprehensive overview of the generic process self-adaptation has been missing. Moreover, faced with the high cost and the difficulty of self-adapting processes, the idea of capitalizing on previous adaptation solutions by implementing reuse techniques is appealing. In this paper, we recommend a hybrid approach to adapt running processes using versions, previous adaptation cases and rules as reuse techniques. Our solution is implemented as an adaptation engine that instantiates the MAPE-K control loop and that can be connected to any BPMN-based BPMS using appropriate adapters. Our adaptation engine is therefore reusable. Finally, we demonstrate the advantages and feasibility of the recommended approach with an example from the crisis domain.

Keywords: Self-Adaptation; MAPE-K; Context; Version; Adaptation Case, Rule.

1 Introduction

The growing complexity and dynamicity of the operating environment of processes emphasizes the need for developing autonomic Business Process Management Systems (BPMS). With the autonomic adaptation of process, it indeed becomes possible to achieve changes occurring in the operating environment of this process with minimal human intervention, cost reduction for performing required adaptation and real-time responsiveness [1]. This is a key advantage for companies that need to quickly and efficiently manage changes to remain competitive. Basically, a fully autonomic BPMS supports an autonomous reasoning to detect needs for adaptation and eventually resolve them. This reasoning is based on the analysis of the current situation of the operating environment and the use conditions of running processes. Autonomic adaptation of running processes is usually achieved using a MAPE-K control loop [2], which is an efficient solution for self-adaptation of systems in autonomic computing. This loop advocates the use of the four components: (i) *Monitor (M)*, which collects data on the managed system and its operating environment, filter the data and aggregates them into symptoms, (ii) *Analyze (A)*, which analyzes the symptoms to detect if changes are required, (iii) *Plan (P)*, which defines the adaptation operations needed to resolve detected changes and (iv) *Execute (E)*, which performs the defined operations to adapt the behavior of the managed system. These components (MAPE) share *Knowledge (K)* between them. The latter involves repositories that contain the neces-

sary information to detect the adaptation needs and resolve them. As this paper deals with the adaptation need resolution issue, it focuses on the *Plan* component.

Process adaptation issue has been highly investigated in recent past. On the one hand, several taxonomies to characterize process adaptation have been proposed in literature. The most suitable one is given in [3]. This taxonomy has identified the following adaptation needs for processes: (i) *adaptation by variability* for handling different models of processes, called variants, each of which is to be used in a specific situation, (ii) *adaptation by evolution* for handling changes in processes, which require occasional or permanent modifications in their models, (iii) *adaptation by deviation* for handling occasional situations or exceptions, which have not been necessarily foreseen in process models, and (iv) *adaptation by looseness* for handling processes, which have unknown or incompletely known at design-time.

In addition to this taxonomy, two main categories of adaptation techniques have been introduced to define how to achieve self-adaptation of processes. The first one, which includes the *goal-based* technique, is based on the definition of a new process fragment based on the current situation of the operating environment and the goal state (desired outputs of the process fragment) when there is an adaptation need. The second one is based on the reuse techniques to be able to adapt a running process by reusing knowledge relating to models or parts of models used in a similar situation. In this paper, we defend the process self-adaptation by reuse techniques because these techniques can reduce the time, cost and effort required to adapt processes, and we have an already such constructed knowledge (*e.g.*, process model versions) in our previous works for adaptation modeling. Generally, there are four reuse techniques proposed in the literature [4]: (i) *rule-based* [5], which reuses a set of pre-defined rules, (ii) *case-based* [6], which enables to reuse historical process adaptation cases (change logs), (iii) *variant-based* [7], which 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 (iv) *version-based* [8, 9], which is an extension of the variant-based as it reuses a set of alternative versions (*i.e.*, variants) or consecutive versions (*i.e.*, evolution of variants) of the process. Each technique may be considered as a guide for defining adaptation operations. Generally, these adaptation operations can impact the different dimensions of a process: (i) its *behavioral* dimension to adapt how it is achieved (process activities, including tasks and sub-processes, and their coordination), (ii) its *organizational* dimension to adapt the resources invoked by task execution, and (iii) its *informational* dimension to adapt the used or produced data by task executions.

On the other hand, several contributions (*e.g.*, [11–14]) have been made to address the process self-adaptation based on the MAPE-K control loop. However, these contributions are incomplete for the following reasons. First, the adaptation techniques of these contributions are either variant-based or goal-based, but none of these contributions combine these techniques in order to find out the best adaptation to perform. Second, they do not consider all process dimensions to ensure comprehensive adaptation of running processes: they focus on the adaptation of only one or two dimensions among the behavioral, informational and organizational ones. Finally, they do not address the issue of autonomic adaptation of processes in a comprehensive and global

approach taking into account all the four adaptation needs identified in the Reichert and Weber's taxonomy [3]: they only take into account one or two adaptation needs but never all four at the same time.

To overcome these weaknesses, we recommend a MAPE-K based adaptation engine for self-adapting running processes to changes occurring in their operating environment. This adaptation engine is designed separately from the BPMS. This separation makes the adaptation engine reusable for various BPMS. In addition, this adaptation engine has the following features. First, it recommends the use of the three following reuse techniques: version-based, case-based and rule-based to improve process adaptation. Second, it advocates a context-based selection of already defined adaptations. In fact, the context notion is used for representing the use condition in which each adaptation operation has to be executed as well as the current situation of the operating environment that influences the execution of processes. Matching both contexts (*i.e.*, use condition of adaptation operation and current situation) allows (i) the detection of the adaptation needs when the operating environment of running process changes and (ii) retrieving the most suitable solution to be adapted for the current situation.

To sum up, the paper deals with the autonomic adaptation of BPMN processes at run-time and more precisely the resolution of detected adaptation needs. Its contributions are as follows. The first one is the recommended adaptation engine architecture, which instantiates the MAPE-K control loop for self-adaptation of processes. The second contribution is the hybrid approach recommended for the *Plan* component, which is responsible for defining the adaptation operations to be carried out in order to resolve adaptation needs. Finally, the feasibility and applicability of the recommended approach is demonstrated by a case study from the crisis domain.

The remainder of the paper is organized as follows. Section 2 provides the state-of-the-art on self-adaptation of processes using the MAPE-K control loop. Section 3 gives an overview of the adaptation engine recommended for self-adaptation of BPMN processes. Section 4 discusses in detail the hybrid approach proposed for realizing the *Plan* component. Section 5 demonstrates the applicability of the proposed approach on the case study. Finally, section 6 summarizes paper contributions and gives some directions for future researches.

2 Related work

In the literature, there are several works that focus on the self-adaptation of processes. The common point of these works is the use of the MAPE-K control loop for detecting adaptation needs and resolving them at run-time. Due to the specific focus of the paper on the adaptation need resolution, this section only considers contributions that really recommend a concrete approach of the related *Plan* component of the MAPE-K control loop.

Ayora *et al.* recommended in [11] a solution that allows variability modeling at design-time by describing process variants using three models: (i) the base model to specify process fragments shared by all process variants (*i.e.*, consistent part of processes), (ii) the variation model to specify the replacement fragments that alternative-

ly can be used for the fragments of the base model, and (iii) the resolution model to specify the context conditions that define the use conditions for the replacement fragments. At run-time, the recommended solution provides the completion of each variation point of the base model with the appropriate alternative fragment that satisfies the context of the operating environment.

In [12], the authors defined an approach that enables the modeling of autonomic processes at design-time and managing them at run-time. At design-time, this approach makes it possible to model all the necessary elements that guide the self-adaptation of a process at run-time: which tasks must be monitored, which context changes impact the execution of the process and how to resolve them. At run-time, this approach uses the MAPE-K control loop for managing autonomic processes. More precisely, it checks all the variation points and examines the context of each variant of these variation points to identify adaptation needs. If adaptations are required, it selects and executes the suitable variant for each variation point, *i.e*., the variant that satisfies the context of the operating environment.

On the other hand, Ferro and Rubira introduced in [13] an adaptation engine for the completion of the loosely parts of ill-defined processes at run-time. This adaptation engine, which is based on the MAPE-K approach, ensures the completion of the loosely part of the process by either (i) selecting existing activities in the process repository or (ii) deriving a new process fragment by analyzing pre-conditions, post-conditions and interdependences between activities.

Finally, Seiger *et al.* proposed in [14] a framework that enables the self-adaptation of processes in cyber-physical systems. This framework allows monitoring and analysis of consistency between the sensed physical world and the assumed cyber world of each task execution. In case an inconsistency is detected, it replaces the resource involved in the task execution by another resource variant and then executes this task.

Table 1 evaluates the previous contributions with respect to the following criteria defined in [15] and related to process self-adaptation:

- *Supervised component:* it identifies the granularity level of adaptation, which can be the process level, the sub-process level and the task level,
- *Adapted process dimension:* it indicates which process dimensions are considered in the examined contribution, which can be the behavioral, the organizational and/or the informational dimensions,
- *Process adaptation needs:* it indicates which adaptation needs of the taxonomy defined by [3] are taken into account,
- *Adaptation technique*: it indicates the technique used to define the needed adaptation operations.

The first observation we can make from Table 1 is that all the examined contributions partially consider the self-adaptation of process dimensions: (i) in [11] and in [13], the adaptation may only impact the behavioral dimension; (ii) in [12], it may impact the informational and organizational dimensions of processes, whereas, in [14], only the organizational dimension of processes may be adapted. This is mainly due to the granularity of the supervision. In fact, (i) when the supervised element is a sub-process or a process, its adaptation may impact the coordination of the process (or sub-process) tasks, and thus the behavioral dimension of processes; (ii) when the su-

pervised element is a task, the process adaptation may impact the resources or the data involved in the task execution, and thus the informational and the organizational dimensions of processes. However, in [14], even if the supervised element is a task, only the organizational dimension of processes can be impacted by the adaptation as the contribution deals with malfunction of resources involved in task realization.

Table 1. Related work evaluation

Works / Criteria	Ayora *et al.*, [11]	Oliveria *et al.*, [12]	Ferro et Rubira [13]	Seiger *et al.*, [14]
Supervised components	sub-process	task	sub-process	task
Adapted process dimensions	behavioral	informational organizational	behavioral	organizational
Process adaptation needs	variability looseness	variability deviation	looseness	deviation
Adaptation technique	variant-based	variant-based	goal-based	variant-based

Second, we can observe that the adaptation needs of Reichert and Weber's taxonomy [3] are partially considered. *Adaptation by looseness* is supported in [11] and in [13], while *adaptation by deviation* is taken into account in [12] and in [14]. As for *adaptation by variability*, it is supported in [11] and in [12] at design-time. However, none of the examined works has dealt with *adaptation by evolution*. Moreover, the adaptation techniques of the examined contributions are either variant-based as in [11, 12, 14] or goal-based as in [13], but none of these contributions has recommended the use of more than one technique in order to find out the best adaptation to perform.

To overcome the limitations of the examined contributions, we recommend an adaptation engine that takes up the interesting features of the examined adaptation engines, namely, the fact to be BPMN-compliant, the MAPE-K control loop, and the context-based approach, but which differs from them in the following respects:

- The proposed adaptation engine considers the process self-adaptation at the following abstraction levels: (i) the *process* and the *sub-process* levels, to support adaptation of the behavioral dimension, and (ii) the *task* level to support adaptation of the informational and organizational dimensions. Considering these three abstraction levels makes the self-adaptation of the three dimensions of processes possible.
- It recommends the mixing of reuse techniques, version-based, case-based and rule-based, in order to improve adaptation and support the different types of process adaptation needs defined in [3].

3 Adaptation Engine Architecture

Fig. 1 below presents the architecture of the adaptation engine we propose to support an instantiation of the MAPE-K control loop. As argued in [2], self-adaptation in the general level encompasses various self-* properties in the major level, including self-configuring, self-healing, self-optimizing and self-protecting. We consider that the approach presented in this paper falls under self-healing category. Self-healing is the

capability of the adaptation engine of discovering, diagnosing and reacting to disruptions [16]. Self-healing can be classified into self-diagnosing and self-repairing, where the former concerns itself with identifying adaptation needs, and the latter focuses on the resolution of the identified adaptation needs, namely the definition and the execution of the adaptation operations. The focus of the proposed adaptation engine is to provide an integrated approach for self-diagnosis and self-repairing using context, versions, adaptation cases and rules.

The Knowledge (K) is composed of the *model repository*, the *instance repository*, the *case repository* and the *rule repository*. The model repository stores versions of tasks, sub-processes and processes as well as their use conditions, described as contexts. The instance repository stores the current situation of the operating environment of running processes, described also as contexts. We characterize the current situation by a set of context parameters defined as pairs (context parameter, value), and the use condition of a version by a set of conditions involving these context parameters. These conditions are defined as triplets (context parameter, operator, value). As for the case repository, it stores a set of all cases representing ad hoc changes defined by the *Plan* component for managing deviations. Each case consists of a past situation that has needed adaptation, described as context and the corresponding solution (*i.e.*, applied adaptation operations). The rule repository stores a set of a priori defined rules by domain experts for managing deviations and dependencies between adaptation operations. Each rule is constructed using *conditions* and *actions* and it has the form: ***if*** *set of conditions* ***then*** *execute actions*.

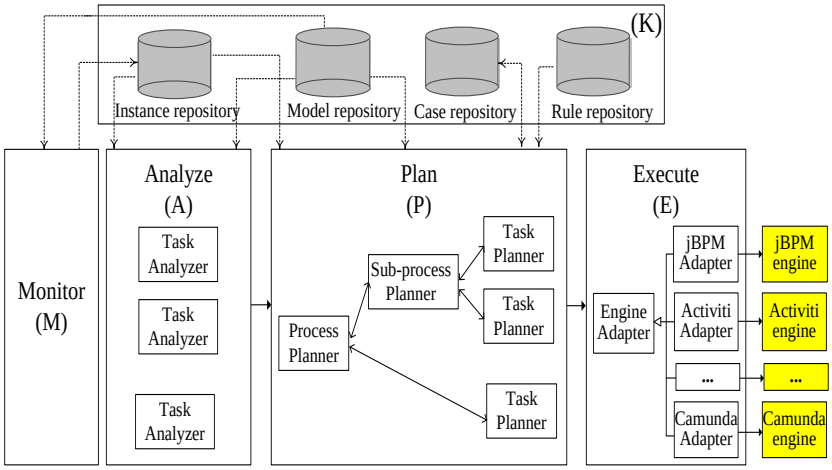


Fig. 1. Adaptation engine architecture

The *Monitor* component implements the *M* of the MAPE-K control loop. It aims at getting an accurate picture of the operating environment of running processes (including their sub-processes and tasks). It receives data from sensors and process engine listeners, filters, aggregates and interprets these data and records them in the *instance repository*.

The *Analyze* component, which implements the *A* of the MAPE-K control loop, is responsible for the detection of the adaptation needs since it generates several *task analyzers* (as many as activated tasks in the process) to compare the context featuring the current situation, described in the *instance repository*, with the use conditions of the versions of each concerned task, described in the *model repository*. Thus, if an adaptation need is detected, the Plan component is triggered. It should be noted that our solution to the *Monitor* and *Analyze* components implementing the self-diagnosing approach is under publication.

The *Plan* component implements the *P* of the MAPE-K control loop. It receives as input the different adaptation needs (when there is at least one task that has identified a need for adaptation) and produces as output an adapted process instance model that meet these adaptation needs. The *Plan* component is composed of several planners that define needed operations to implement the identified adaptation: one process planner, as many sub-process planners as sub-processes in the supervised process instance and as many task planners as tasks in the supervised process instance. Each planner deals with its own adaptation and defines the operations required to carry it out by reusing defined solutions stored in the *knowledge* base. Thus, a *process planner* has a better global vision of the adaptation needs and defines the adaptation operations for the process and its components (sub-processes, tasks, resources and data), while at the lower level, *i.e.*, a task or a sub-process planner solves a local adaptation need at the concerned activity (task or sub-process) when there is no adaptation operations defined by the process planner. Notifications between planners are visualized in Fig. 1 as thin arrows. More details about the *Plan* component are presented in Section 4. It should be noted that the dotted arrows of Fig. 1 visualize the read/write operations in K of the *Monitor*, *Analyze*, and *Plan* components.

Finally, the *Execute* component implements the *E* of the MAPE-K control loop. It (i) receives from the *Plan* component the adapted process instance model, (ii) generates the generic operations to be carried out to migrate the considered process instance model to the adapted one, (iii) maps these operations into operations of the target process engine, since each process engine has different operation signatures and ways of performing each same adaptation operation, and (iv) performs them by invoking the process engine by means of a native Application Programming Interface (API). Since each process engine (*e.g.*, Activiti, Camunda, jBPM) has its own specificities, we have defined several engine adapters, such as *jBPM Adapter*. Each engine adapter is suitable for a process engine; it has to map the operations defined by the *Plan* component onto operations that can be understood by this process engine. Ultimately the *Execute* component triggers the execution of these mapped adaptation operations in the target process engine. We do not detail this component implementation in this paper as we adopt the *access layer* of the academic generic BPMS proposed in [17], that supports communications with the target process engines allowing the realization of the generic operations mapped in concrete ones, and maps the generic operations to operations that conform to the target process engine. For more details about this generic BPMS implementation, the reader can refer to [17] and [18].

We detail below the recommended approach for defining adaptation operations required for resolving adaptation needs.

4 Detailing the recommended Approach for Adaptation definition

This section details the hybrid approach recommended for the definition of adaptation operations by the *Plan* component to resolve process adaptation needs. This approach recommends the use (i) of versions, previous adaptation cases and rules for adaptation achievement and (ii) context-based selection of versions as well as previous adaptation cases. As shown in Fig. 2, once an adaptation need is received from the *Analyze* component, three major steps are possibly performed.

The first step "**Search compliant process model version using contexts**" consists of querying the *model repository*, which contains the process model versions and their use contexts, with the aim of retrieving the process model version satisfying the current situation of the operating environment. More precisely, it searches the appropriate process model version by comparing the current situation that provokes need for adaptation and that is defined in the *instance repository*, with the use conditions of the versions of this process instance, defined in the *model repository*. More precisely, this step goes through the following actions. The first action is dedicated to the identification of the possible model versions of the considered process. It is implemented as queries on the *model repository*. The second action calculates, for each identified version, the similarity between the current situation and their use condition. This action is implemented by Algorithm 1, which takes as input the set of versions identified in the previous action and the current situation described in the *instance repository*. It uses the following functions:

- **getUseCondition (v)** returns the use condition of the version *v*,
- **checkCondition (c, cs)** returns *true* if the condition *c* is verified in the current situation cs, otherwise *false*,
- **add (v)** adds the version *v* to the set of versions to be returned,
- **card (uc)** returns the number of conditions involved in the use condition *uc*.

Three scenarios can occur:

- **Scenario 1.1-** There is exactly one process model version that satisfies the current situation, *i.e*., the current situation acquired from the *instance repository* exactly matches the use condition of the returned version. So in this case, the **check state-related compliance** is performed for verifying if the considered process instance is compliant with the returned process model version or it is not compliant. This allows ensuring that the resulting execution states of the considered process instance also remain correct and consistent states in the returned version. The goal of this check is to ensure the correctness of process instance migration to the returned process model version. Moreover, the state-related compliance check is done according to the state compliance conditions proposed in [19], which allow checking if a process instance model version (*Si*) is compliant with a process model (*S'*). In case a state-related incompliance is detected, *i.e*., it is not possible to migrate the running process instance to the returned process model version, the *process planner* searches for a suitable previous adaptation in the *case repository*.

- **Scenario 1.2-** There are many retrieved process model versions satisfying the current situation. In this case, the *process planner* determines the proximity between the process instance model version and each returned process model version using the proximity calculation algorithm proposed in [20]. The goal of this proximity calculation is to order the returned process model versions based on their proximity to the process instance model version. Afterward, the *process planner* checks the state-related compliance of this version with the ordered process model versions, starting with the closer process model version to this instance model version, to find a process model version that satisfies the state-related compliance conditions. When there are no process model versions ensuring the state-related compliance, then the search of suitable adaptation cases in the *case repository* is triggered.
- **Scenario 1.3-** There is no suitable process model version to the current situation. Let us remember that to consider a model version suitable, each condition involved in the use condition of this version must satisfy the current situation. So in this scenario, the step "**Search compliant adaptation cases using contexts**" is performed for reusing previous adaptations used in a similar situation and defined in the *case repository*.

Algorithm 1: Similarity Calculation

```
Function SimilarityCalculation (versions: set (Version),
cs: Current Situation): set (Version)
Local
sim: Integer, rvs: set (Version) = ∅, v: Version, c: Con-
dition,
uc: set (Condition)
Begin
   For each v in versions
     uc = getUseCondition (v)
     sim = 0
     For each c in uc
       If checkCondition (c, cs) Then
         sim ++
       End If
     End For
     If (sim/card (uc) = =1) Then
       rvs.add(v)
     End If
   End For
   Return rvs
End Function
```

The second step "**Search compliant adaptation cases using contexts**" is triggered when either (i) there is no process model version satisfying the state-related compliance from the returned ones in the first or the second scenario, or (ii) there is no suitable process model version returned in the third scenario of the first step. In this second step, the *process planner* looks for previously defined adaptation cases in similar situation by the lower level planners (planners of tasks and/or sub-processes), in the *case repository*. More precisely, it searches to retrieve the most relevant adaptation cases from the *case repository* by comparing the current situation of the considered

process instance with the situation of the adaptation cases, as well as the process instance model version with the process model versions of the adaptation cases. Two possible scenarios can then occur:

- **Scenario 2.1-** There is no retrieved adaptation case corresponding to the model version of the current process instance and satisfying the current situation of the operating environment. So, in this case, the rule-based adaptation is triggered to define a new solution resolving the need for adaptation.
- **Scenario 2.2-** There is one or more retrieved adaptation cases corresponding to the model version of the current process instance and satisfying the current situation. As indicated in Algorithm 2, these adaptation cases are then examined one by one. For each of them, the defined adaptation operations in the considered adaptation case are applied to the process instance model version in order to adapt it. Then, the **check state-related compliance** is triggered to check the state-related compliance of the process instance model version with the resulting one. When there is no resulting process model version verifying the state-related compliance conditions, the rule-based adaptation is performed to define a new solution resolving the need for adaptation; otherwise, the resulting process model version is to approve by a domain expert.

Algorithm 2: adaptation Cases Application

```
Procedure adaptationCasesApplication (cases: set (Adaptation
case), pi: Process instance model version)
Local
c: Adaptation case; V', Vtemp: Process model version
Begin
  V' = pi
  For each c in cases
    /* Calculate the new process instance model version Vtemp=
    V'+ Δcase */
    Stemp = generateAdaptedmodel (V', c)
    // Checks state-related compliance
    CpRes= stateComplianceChecks (pi, Vtemp)
    If (CpRes is Compliant) Then
      V' = Vtemp
    End If
  End For
  If (V' = = pi) Then
    /* Case of the process instance is not compliant with all
    retrieved adaptation cases */
    Rule-basedAdaptation ()
  Else
    adaptationApprovement (V')
  End If
End Procedure
```

The third step "**Define new adaptation case using rules and contexts**" is triggered when there is no solution defined as a version and no adaptation case that deals with the adaptation of the considered process instance. This step uses the a priori defined adaptation rules stored in the *rule repository* and the current situation described in the *instance repository* to define a new adaptation case that resolves the identified

adaptation needs and accordingly stores it in the *case repository*. Two main types of adaptation rules are supported: (i) the *reactive rules*, which react to a given situation, and (ii) the *dependency rules*, which manage the impact of adaptation operations as indicated in [21]. It should be noted that both types of adaptation rules define adaptation operations using either adaptation patterns or primitives like suspending or redoing a task version. This step is as follows. First, the *process planner* creates a planner per task that has identified a need for adaptation. Each *task planner* deals with the definition of the adaptation operations using reactive rules for resolving the local adaptation need. Once the adaptation operations are defined, they are forwarded to the *planner* of the component in which the considered task is involved, and which is either a *sub-process planner* or a *process planner*. This latter has to analyze the dependency of the defined adaptation operations using dependency rules. After that, the adaptation operations are suggested to a domain expert, who will approve the solution or further adapt it. Finally the solution (*i.e.*, process instance model version, the approved adaptation operations and their situations) will be stored by the *process planner* in the *case repository* as a new adaptation case that can be reused in a situation similar to the current one.

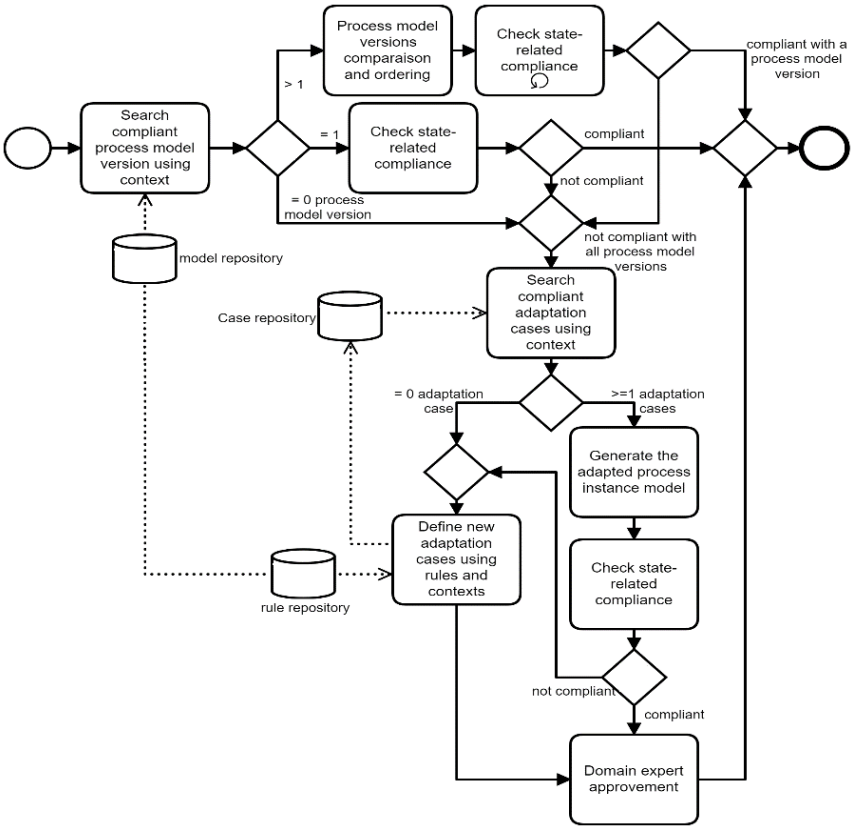


Fig. 2. Process instance adaptation definition steps

Note that we use the version-based technique at the beginning for the following reasons. First, this technique always guarantees the process model correctness, because all the process model versions are verified and validated by a process designer at design-time. Second, in the version-based technique, the process instance is always compliant to a process model version, which is stored in the model repository.

5 Case study: the flood management process

This section illustrates the applicability of the recommended approach through the case study *Flood Management Process* (FMP). First it shows how to model at design-time the required knowledge for flood management process adaptation using versions and rules. Then it uses this case study to demonstrate the execution of a sample adaptation.

5.1 Rules and versions modeling at design-time

This sub-section introduces the FMP case study, which is a simplification of a more complete case study in the context of the flooding of a major French river "the Loire" on the city of Blois, and more particularly on the district of Mareuil, which is protected by a dyke. This district is the object of a very particular attention at the time of floods of the Loire River: a rise in water levels caused by heavy rainfall upstream, or on the affluent of the Loire, may cause important damages on the potentially impacted area.

Like any other process, the FMP might be subject to different operating environment variations that could interrupt its functioning. Indeed, changes relative to the water level, precipitation amount, lack of resources, impacted roads, etc., impose the process to self-adapt to take into account these changes. For this reason, we have defined several versions of the model of this process and of its components, namely several versions of its tasks and sub-processes, one per situation. The first version defines the process of watching over the state of the dike while the second defines the process of lowering the water level of the Loire by opening the Boullie spillway upstream. In addition, the three other versions, which are triggered when the situation is of serious concern, define what to do when an evacuation decision is made. According to the crisis cell, these five process model versions depend upon the following context parameters among others:

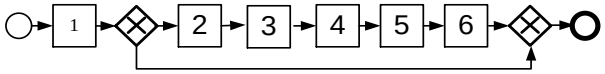
- **Water level,** which indicates the level of water rising in the Loire,
- **Impacted area,** which features the size of the population potentially impacted,
- **Road state,** which can be *not flooded*, *flooded and drivable* or *flooded and blocked*. When the roads are not flooded, so peoples evacuate themselves when the water level is not above 4,5 m; otherwise, when the roads are flooded but still drivable, so the people evacuation is carried out using specific vehicles with help of gendarmes. While, when the roads are blocked because they are highly flooded, people evacuation must be carried out by firefighters with zodiacs.
- **water growth,** which indicates the rapidity of water rising.

Table 2 shows these different versions and their corresponding use conditions (*i.e.*, in which situation condition the process model version must be used). The different conditions defined in the use condition of each process model version are connected to each other by the logical operator "*and*".

Table 2. Use conditions for flood management process model versions

Version id	Version Use condition			
	Water level	Impacted area	Road state	Water growth
FMP.V1	< 2	= "Urbanized"	= "Not flooded"	= "Slow" or = "Moderately fast"
FMP.V2	>=2 and < 3	= "Urbanized"	= "Not flooded"	= "Slow" or = "Moderately fast"
FMP.V3	>=3 and <= 4,5	= "Urbanized"	= "Not flooded"	= "Slow" or = "Moderately fast"
FMP.V4	> 4,5	= "Urbanized"	= "Flooded and drivable"	= "Slow" or = "Moderately fast"
FMP.V5	> 4,5	= "Urbanized"	= "Flooded and blocked"	= "Slow" or = "Moderately fast"

Due to lack of space, we explain below only the third model version of this process among the five ones. As shown in Fig. 3, this version is triggered in urbanized impacted area when the water level of the river "Loire" in France rises above 3 m and the roads state is not flooded. In response to this event, the crisis cell decides whether or not to evacuate people from the flooded zones by assessing the flooding situation. In case an evacuation is needed, the Prefect emits an evacuation order, then the COD, which is the operational committee set up within the crisis cell, informs the population about the flood. After that the gendarmes proceed to the evacuation of people from the flooded zones using vehicles. Finally, the crisis cell reports on the evacuation, and the Prefect sends the report to the interior ministry.



1: Decision making for evacuation (DET)
2: Emit evacuation order (EOT)
3: Inform population (IP)
4: People evacuation (PET)
5: Report on evacuation (ERT)
6: Send evacuation report to the interior ministry (SERT)

Fig. 3. The third model version of the flood management process "FMP.V3"

Moreover, we define for each process component (process, sub-process or task) rules, that manage component deviation, and their dependency. We give below two rules related to *People evacuation* task in the form of if-then statements. The first rule is used when the road state is not flooded and the water keeps rising very fast, so there is a risk that the road state will be flooded. In this case, people's evacuation must be carried out by gendarmes with specific vehicles; it adds resources (*i.e.*, gendarmes) to the *People evacuation* task. While the second rule is used to manage dependency of the people evacuation. For instance, when the *People evacuation* task is carried out by

gendarmes, then it is necessary to insert the new task *Evacuation supervision* that allows the supervision of peoples during the evacuation. It should be noted that these rules are modeled using Drools rule engine.

R1: if Water_growth = "Very fast" Then AddTaskResource (Running process, People evacuation task, gendarme).

R2: if AddTaskResource (Running process, People evacuation task, gendarme) Then InsertSerialTask (Running process, Evacuation supervision task, People evacuation task, Report on evacuation).

5.2 Process self-adaptation at run-time

The goal of autonomic adaptation of processes is to modify the process instance model version in response to changes in its operating environment. To demonstrate this, we suppose that the third model version of the flood management process (FMP.V3 from Table 2) is running for Mareuil district affected by Loire's floods and that the *people evacuation* task is activated. For this illustration, we refer to the following context parameters and values, which feature the current situation of the operating environment: *Water level = 4 m* and *Impacted area = "Urbanized"* and *Road state = "Not flooded"* and *water growth = "Very fast"*. In response to this situation, the *Plan* component, precisely the *process planner* of the running process model version proceeds to the "Search compliant model version using contexts" step. First it identifies the corresponding model versions and their use conditions, second it calculates, for each of these model versions, the similarity between the model version use condition and the current situation. The result of this step is that there is no existing version that deals with the current situation. Given that there is no solution modeled as a model version, the *process planner* performs the "Search compliant adaptation cases using contexts" step, which also returns that there is no stored adaptation case that addressed the current situation. So in this case, the *task planner* of the *People evacuation* task is triggered to search rules that match with the current situation. The result of this search is the rule R1, which allocates gendarmes to the activated model version of the *People evacuation* task. Then the *process planner* is triggered to manage dependency of this change using rule R2. The final result of this adaptation is illustrated in Fig.4.

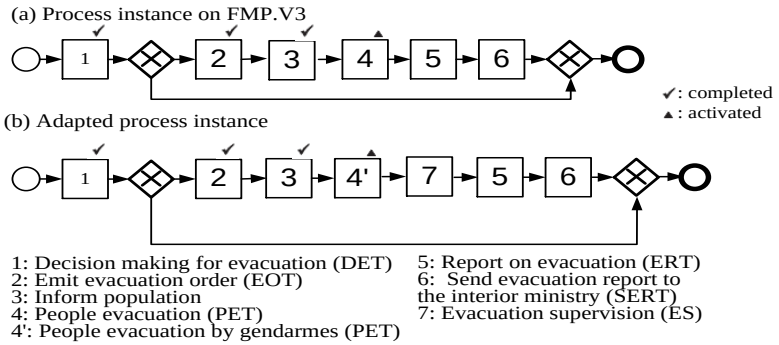


Fig. 4. Adapted process instance

6 Conclusion and Future Work

Adapting processes to the frequent changes of their operating environment remains a challenging and complex task. To tackle this issue, this paper recommends an Adaptation Engine based on the MAPE-K approach of autonomic computing so that supervised processes self-adapt to changes occurring in their operating environment. More precisely, the paper focuses on the definition of the adaptation operations required to resolve adaptation needs using contexts and reuse techniques: version-based, case-based and rule-based.

Benefits of our solution are as follows. First the recommended solution benefits from the MAPE-K control loop advantages. This loop from the autonomic computing field makes possible the implementation of adaptation for monitored processes with minimal human intervention. Second, the separation between the adaptation engine and the process engine with which it interacts makes the adaptation engine reusable for various process engines (BPMS). Third, the combined use of version-based, case-based and rule-based techniques makes possible to take the different adaptation needs identified in Reichert and Weber's taxonomy for well-defined processes into account. Thus our recommended solution supports in a coherent framework the adaptation by deviation, evolution, variability and looseness. Third, adaptation of processes at the three abstraction levels makes the self-adaptation of all the dimensions of processes possible. Finally, the use of several adapters for connecting process engines aims to overcome the drawbacks of embedding the adaptation logic within the process engine, thus it improves the reusability and independency of the adaptation engine.

As future works, we plan to continue improving this adaptation engine in two main directions:

- Improvement by performing an analysis of the impact of the defined adaptations by the *Plan* component before they are transmitted to the *Execute* component. In other words, we need to assess the quality of the adapted process instance model version in terms of comprehensibility and modifiability such as in [22].
- Improvement by considering the collaborative aspects of the adaptation engine, as BPMN allows the definition of collaborative processes within collaboration and choreography diagrams.

References

1. J. Oukharijane, I. Ben Said, M. A. Chaâbane, E. Andonoff, and R. Bouaziz, "Towards a New Adaptation Engine for Self-Adaptation of BPMN Processes Instances", in *14th International Conference on Evaluation of Novel Approaches to Software Engineering*, 2019, pp. 218–225.
2. IBM, "An architectural blueprint for autonomic computing", *IBM White Paper*, vol. 31, 2006.
3. M. Reichert and B. Weber, "Enabling flexibility in process-aware information systems: challenges, methods, technologies", *Springer Science & Business Media*, 2012.
4. M. Fantinato, M.B.F.d. Toledo, L.H. Thom, I.M.d.S. Gimenes, R.d.S. Rocha, and D.Z.G. Garcia, "A survey on reuse in the business process management domain", *International Journal Business Process Integration and Management*, vol. 6, no. 1, pp. 52–76, 2012.

5. R. Müller, U. Greiner, and E. Rahm, “Agentwork: a workflow system supporting rule-based workflow adaptation”, *Data Knowledge Engineering*, vol. 51, no. 2, pp. 223–256, 2004.
6. M. Minor, R. Bergmann, and S. Görg, “Case-based adaptation of workflows”, *Information Systems*, vol. 40, pp. 142–152, 2014.
7. F. Milani, M. Dumas, N. Ahmed, and R. Matulevičius, “Modelling families of business process variants: a decomposition driven method”, *Information Systems*, vol. 56, pp. 55–72, 2016.
8. F. Ellouze, M. A. Chaâbane, E. Andonoff, and R. Bouaziz, “Onto-VP2M: A New Approach to Model and Manage Collaborative Process Versions using Contexts and Ontologies”, *International Journal e-Collaboration*, vol. 13, no. 3, pp. 39–62, 2017.
9. I. Ben Said, M. A. Chaâbane, E. Andonoff, and R. Bouaziz, “BPMN4VC-modeller: easy-handling of versions of collaborative processes using adaptation patterns”, *International Journal of Information System and Change Management*, vol. 10, no. 2, pp. 140–189, 2018.
10. H. Ariouat, E. Andonoff, and C. Hanachi, “From Declarative Knowledge to Process-based Crisis Resolution: application to Flood Management”, in *Hawaii International Conference on System Sciences*, 2019, pp. 1–10.
11. C. Ayora, V. Torres, V. Pelechano, and G. H. Alférez, “Applying CVL to business process variability management”, in *VARiability for You Workshop: Variability Modeling Made Useful for Everyone*, 2012, pp. 26–31.
12. K. Oliveira, J. Castro, S. España, and O. Pastor, “Multi-level autonomic business process management”, in *International Conference on Enterprise, Business-Process and Information Systems Modeling*, 2013, pp. 184–198.
13. S. Ferro and C. Rubira, “An architecture for dynamic self-adaptation in workflows”, in *International Conference on Software Engineering Research and Practice (SERP)*, 2015, pp. 35–41.
14. R. Seiger, S. Huber, P. Heisig, and U. Assmann, “Enabling Self-adaptive Workflows for Cyber-physical Systems”, *Software and System Modeling*, vol. 18, no. 2, pp. 1117–1134, 2019.
15. J. Oukharijane, I. Ben Said, M. A. Chaâbane, R. Bouaziz, and E. Andonoff, “A Survey of Self-Adaptive Business Processes,” in *32nd International Business Information Management Conference*, 2018, pp. 1388–1403.
16. M. Salehie and L. Tahvildari, “Self-adaptive software: Landscape and research challenges”, *ACM transactions on autonomous and adaptive systems*, vol. 4, no. 2, pp. 1–42, 2009.
17. A. Delgado and D. Calegari, “A generic BPMS user portal for business processes execution interoperability”, in *Latin American Computer Conference*, 2019, pp. 1–10.
18. D. Rodriguez, B. Remedi, and A. Guggeri, “Generic BPMS user portal”, 2018. https://gitlab.fing.edu.uy/opencoal/portalbpms.
19. S. Rinderle, “Schema evolution in process management systems”, Thesis, Université de Ulm, 2004.
20. M. A. Châabane, “De la modélisation à la spécification des processus flexibles : Une approche basée sur les versions”, Thesis, Université Toulouse 1, Septembre 2012.
21. M. O. Kherbouche, “Contribution à la gestion de l'évolution des processus métiers”, Thesis, Université du Littoral Côté d'Opale, 2013.
22. J. Oukharijane, F. Yahya, K. Boukadi, and H. Ben-Abdallah, “Towards an approach for the evaluation of the quality of business process models”, in *International Conference on Computer Systems and Applications*, 2018.