Adaptative Workflows for Process Management

Charbel Kady, Francois Trousset, Nicolas Daclin, Grégory Zacharewicz, Charles Yaacoub, Adib Akl

Adaptative Workflows for Process Management

Charbel KADY, Francois TROUSSET, Nicolas DACLIN, Gregory ZACHAREWICZ, Charles YAACOUB, Adib AKL

Abstract In industry, modeling is used to enhance productivity, facilitate analysis, and increase profitability. Modeling is used to represent the workflow activities of a prescribed business process operation. But what happens if an unexpected event occurs? This might lead to a deviation of prescribed business operation. In related approaches all solutions are predefined in a way or in another where in other words, most events must be predicted in advance and the others are not caught. This study will introduce a new approach to consider unexpected events occurring in workflow execution. Then, after characterizing the situation, the proposed approach will search a library for compatible solutions to repair the model. Next, the best solution is selected (automatically or manually) and injected in the workflow with respect to an objective function set by experts. Moreover, the library of models can be open to external contribution and benefits from collaborative works.

1 Introduction

Nowadays, the Business Process (BP) is becoming more and more complex in organizations. It could encompass thousands of activities to be achieved. Consequently, these BPs need to be modeled to control their complexity. In case of unexpected malfunction, the Business Process will deviate from the prescribed model. In industry, to overcome these situations, the habit has been taken to over-specify the models to cope with every foreseen situation. But unfortunately, it is impossible to describe all situations that could occur (the recent COVID-19 situation is a good example of this problem). In this case there is an urgent need to repair the model for the sake of sustaining the company's operation where the following question is raised: "how to model the response to this problem?". In fact, it is costly to imagine all kind of malfunctions that could happen during runtime and propose exhaustive repair for each problem. On one hand, in literature, a main model is proposed, and many variations of the same model could be considered

Charbel KADY[1,2]
[1]School of Engineering, Lebanese American University (LAU), P.O. Box 36, Byblos 1401, Lebanon
charbel.kadi@lau.edu.lb, charbel.kady@mines-ales.fr

Francois TROUSSET, Nicolas DACLIN, Gregory ZACHAREWICZ
[2]IMT Mines Ales, LSR, 6 avenue de Clavières, 30319 Alès, France
{gregory.zacharewicz, nicolas.daclin, francois.trousset}@mines-ales.fr

Charles YAACOUB, Adib AKL
[3]School of Engineering, Holy Spirit University of Kaslik (USEK), Jounieh P.O. Box 446, Jounieh 1200, Lebanon
{charlesyaacoub, adibakl}@usek.edu.lb

during runtime. On the other hand, the approach described in this paper will not try to model all cases. Instead, a main model will be built with the help of domain experts and when the actual business process deviates from the main model during execution (expected or unexpected deviation), the system will search a library for compatible solutions to repair the deviation in the model, evaluate them and select (automatically or manually) the best one to be injected in the main model. The library initially will be populated with basic blocks (e.g. AND fork), and it with time the users enrich this collaborative library with workflow blocks that combines one or several activities which can serve as solution for certain deviations. Decision-maker considerations may be carried by an objective function to classify solutions. In addition, the mentioned library can reach outside contributors and benefit from collaborative efforts to enrich the solutions repository.

This paper is organized as follows: section 2 presents a state of the art, the proposed methodology is described in section 3, and conclusions are drawn in section 4 with future work perspectives.

2 State of the Art and related work

Previous studies treated fully or partially four main aspects in relation with the main problem statement: *Problem Identification, Solution Evaluation, Model Validation* and *Model Reparation.* The first aspect "*Problem Identification"* is the subject of many studies and groups such as the *One-Way* project [1] where a *Numerical Twin* is defined to detect deviation of the system with respect to its prescribed behavior. The second aspect is "*Solution Evaluation*" which was also addressed by many studies as (Ducq *et al.)* [2] where performance indicators are defined on models and methods of aggregation. These performance indicators were used for example to evaluate interoperability of systems (Heguy) [3] and to aggregate performance indicators (***PIs***) computed on single BPMN elements to parts of the model (Ougaabal) [4]. The third aspect, "*Model Evaluation*", where in this regard, several theories have been proposed to validate a model. For instance, the work of (Kherbouche) [5] who contributed to this direction by proposing a method to validate the modification in workflow resulting from destroying and inserting tasks based on *PIs.* (Mallek-Daclin) [6] suggested verification techniques to validate interoperability in a collaborative process model based on the data quality and time.

As for *"Model Reparation"* two main categories can be identified: "*Over Specified Models*" (OSM) and "*Under Specified Models*" (USM). The first category (OSM) emphasizes the variability approach, where all possible solutions are predefined in a way or another and the system's response will be defined at runtime with respect to a variation point and conditions. Some of these approaches used the concept of object-oriented languages to code workflows and handle possible deviations (Alex Brogida et al. [7]). (Svendsen) [8] used the Common Variability Language (CVL) to describe and generate variants of the same model to achieve

applications reconfiguration while (Honghao et al.) [9] discussed the workflow reconfiguration. In his survey, (La Rosa et. al.) [10] defined four different types of approaches in variability where in each one a different variation point is considered: 1) "Node Configuration" where the node is considered as the point of variation, and for all node, different paths exist (Van der Aalst et. al.). [11] "Element Annotation" where domain properties are assigned via Boolean expressions and selection can be made manually or by aided model; in other words, a test on a specific condition is performed on the level of each task (*e.g., less cost*) (Becker) [12]. 3) "Activity Specialization", named as such to stress that a custom-made business model relies on the specialization on the activity level (Bayer et. al.) [13], and finally, 4) "Fragment Customization" which depends on constraints; an activity is added if it satisfies a given constraint or rejected if it does not (Hallerbach et. al.) [14]. In the above OSM approaches and in either deviation or reparation modes, models rely on the idea that all paths should be prescribed in advance in a way or in another, yet, in most of real-life situations, it is proven to be impossible to predict all situations [15].

The second category (USM) focused on management by exception, where a main model is set for normal operation and an exception handling mechanism is set to treat unexpected problems that could occur on the model at runtime using a library of sub-models. (Nick Russel et al.) [16] define a categorization of exceptions that could occur during execution of a process model and methods to handle them. (Michael Adams et al.) [17] maintained a database of exception handling processes (called *exlets*) that could respond to predefined categories of exceptions. The database may be constructed statically during design or dynamically at runtime. A recent study by (Jasinski et al.) [18] proposed a workflow management system that manages a controlled environment using dynamically produced workflows. Exception detection and handling in process creation generates mitigation recommendations for potential occurrences. It enables the rapid formulation of new tasks, both known and unknown, as well as the evaluation of the quality of the created recommendation via input from the managed environment. (Kerstin et al.) [19] pushed the exception handling even further by introducing an extra layer of fragments that can be reused in workflows during failure. The limitation in the existing USM approaches is that the *Reparation* is still local. That is, in case of malfunction, the main workflow will stop when an exception occurs, will select a solution from a library and will continue from the point where the problem has been identified.

In this manuscript, the proposed solution consists of two parts. The first, is a primary model that will be developed with the assistance of domain experts using the OSM approach. While in contrast the second is based on USM approach where a collaborative library is built progressively, and users will gradually enrich it with workflow blocks that combine one or more activities and can serve as a solution for certain deviations. These solutions are evaluated while the best one is selected and injected in the main model to correct the deviation. Based on all the above, and in either OSM or USM approaches, *Reparation* is poorly considered and remains

based on simplified conditions, does not rely in general on human decision and does not cover the collaborative aspect of a solution.

3 Methodology and proposed solution

During a workflow execution, three types of problems (expected or unexpected) are recognized: the first type of problems could be identified at the entry level of a specific task, the second when a task fails to execute due to lack of resources or due to constraints, and finally the third at the exit when the task is generating less than expected for the system to perform normally. When one of these problems occurs, the system will search for a possible compatible solution from a library of small workflows (sub-models) that could help in repairing the workflow, taking into consideration the needed resources to this correction and the constraints for this solution to be chosen. The selected solution could either 1) correct the entry of a task, 2) restore or bypass a defective task or 3) correct the output of the task to repair the workflow. In the case of a bypass, and knowing the solution entry point, the system will evaluate all possible exit points on the workflow depending on an objective function set by a domain expert. Then, all compatible solutions are evaluated (*Correction, Repair or Bypass*) and sorted. As for the decision, it could be automatic where the system will choose the solution with the highest rank, or in other case, one or more solutions are presented to the decision-maker (human operator) to make the choice and to help him to decide.

Still to consider the situation where none of the solutions is selected due to incompatibility, constraints, or lack of resources, a manual reparation is proposed to the operator, and a new solution is added to the collaborative library where it could contribute to solving similar problems in the future. To be able to achieve the above some needed system information will be defined (in paragraph 3.1) and the methodology will be fully discussed starting from (paragraph 3.2).

3.1 Definitions and Notations

Before proceeding with the methodology for this new approach, there is a need to define some terms to be able to answer the questions raised in the introduction (Section 1). Therefore, a set of definitions and notations are recalled from literature while others are introduced to describe the proposed solution. For that purpose, Business Process Modeling and Notation (BPMN) 2.0 [20] is used, where a task is modeled using a rectangle, the starting event as a circle and a bold circle to represent the ending event of a model (e.g., Fig. 1.Simple Task A).

Characterization of the system state: As per the definition of Derek Rowell [21], a dynamic system state is a minimum collection of variables that can define a system and its behavior at a specific time. In the rest of this manuscript, the real system

state at the entry of an activity $\boldsymbol{A}$ will be annotated as a vector $\widehat{\boldsymbol{I_A}}$ while the real system state at the exit of an activity $\boldsymbol{A}$ will be annotated as a vector $\widehat{\boldsymbol{O_A}}$. A system state could be a set of variables that incorporate internet of things (IoT) parameters (e.g., temperature, weight, humidity), resources or any other parameter.

Characterization of a task (Activity/Solution). In normal operation, a smooth transition can be observed from one activity to another depending on the system state. But, in case of malfunction, at any moment on the main workflow, the system will stop the execution and search for compatible solutions from a library of patterns. Activities and solutions are both tasks and characterized by 1) an input vector $\boldsymbol{I}$ that describes the expected state variables to enter a task (activity) $\boldsymbol{A}$ and it will be annotated by $\boldsymbol{I_A}$, while 2) the output vector $\boldsymbol{O}$ will describe the state variables produced after executing the task and it will be annotated by $\boldsymbol{O_A}$ 3) the required resources vector to achieve a task $\boldsymbol{T}$ will be annotated by $\boldsymbol{R_A}$, and finally 4) the constraints vector $\boldsymbol{C}$ which is a set of necessary conditions to allow the execution of a task and will be annotated by $\boldsymbol{C_A}$.

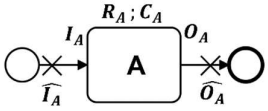


Fig. 1. Simple Task A

Performance Indicators (PIs) and Objective Function (OF): In order to evaluate each solution from the library it is necessary to set an objective function (***OF)*** that evaluates and ranks every possible compatible solution with respect to several performance indicators (***PIs***). An expert in the domain will choose the best objective function that fulfills his goals. For example, in voice over IP industry, time delay, reliability and availability could be used as performance indicators while in other domains, like aviation for instance, in addition to the previously mentioned ***PIs***, quality, security and cost could be used as well. As for the aggregation of each performance indicator, it may differ from an indicator to another and for the rest of this manuscript as per the state of the art, the work of Yves Ducq [2] is considered, where all activities can be reduced into three main categories: Sequential reduction, "Or" reduction and "And" reduction. Performance indicators $\boldsymbol{PI}$s and their number could vary from industry to another, and the set of performance indicators is annotated by $\boldsymbol{P} = \{\boldsymbol{PI_1}, \boldsymbol{PI_2}, \ldots . \boldsymbol{PI_n}\}$.

Later (in paragraph 4.2.5), the Objective Function concept is presented along with an example to better illustrate the point.

3.2 Problem identification: "What kind of problems could occur?"

In normal operation and in order to execute an activity $\boldsymbol{A}$, the following should be satisfied: 1) All required resources $\boldsymbol{R_A}$ for executing task A should be available

2) all constraints $\boldsymbol{C_A}$ are verified and finally 3) the system state at the entry of $\boldsymbol{A}$ should be equal or better than expected for $\boldsymbol{A}$ to execute:

$$\boldsymbol{I_A \preccurlyeq \widehat{I_A}}$$

Moreover, 4) the output of the task should be better or equal to the System state at the exit point:

$$\boldsymbol{O_A \succcurlyeq \widehat{O_A}}$$

In contrast with normal operation, while executing the above simple task $\boldsymbol{A}$, three types of problems could occur: the first where the state of the system at the entry is less than expected for task $\boldsymbol{A}$ to be executed, the second case would be that $\boldsymbol{A}$ itself is down and the third and last possible case is that the output after executing the task will correspond no longer to the expected output of the activity $\boldsymbol{A}$ or the system state at the exit.

First, as a start the atomic case (cf. Fig. 1.Simple Task A) is considered and later it can be applied on any complex workflow. In fact, the output of an activity $\boldsymbol{A}$ could be considered as an input of another activity $\boldsymbol{B}$ and so on.

In summary, all problems can be reduced into three cases which can be applied along the workflow:

***1) Input Problem*: A problem at $\boldsymbol{\widehat{I_A}}$ level (PS)**

$\boldsymbol{A}$ cannot be executed as it is not receiving the proper set of state variables. The system state at the entry $\boldsymbol{\widehat{I_A}}$ does not match with what is expected for $\boldsymbol{I_A}$ and it will be annotated by $\boldsymbol{\widehat{I_A} \prec I_A}$.

In this case a problem is detected at the input of task $\boldsymbol{A}$.

***2) Failing Task*: Task "*A*" is failing to execute (PA)**

In this case we have no problem at the system state level and a failure is occurring during the executing the task. This Failure could be due to lack of resources (annotated by $\boldsymbol{\widehat{I_A} \prec R_A}$) or due to a specific constraint.

***3) Output Problem*: A problem at the output of the Task (PO)**

In this case, the output of activity $\boldsymbol{A}$ is incompatible with the system state $\boldsymbol{\widehat{O_A}}$. In other words, the task $\boldsymbol{A}$ is producing less than expected for $\boldsymbol{\widehat{O_A}}$and will be annotated as: $\boldsymbol{O_A \prec \widehat{O_A}}$.

3.3 Repairing typology: "How to repair the model?"

In this manuscript three simple types of repairs are considered: first, the *Correction (entry/exit)* where the system state at the entry or the exit is corrected. Second, the *Restore* of failing task $\boldsymbol{A}$ and finally the replacement of one or multiple activities, the *Bypass*.

*** When a problem is identified at the entry-level:** In this case the state the system state at the entry is less than expected for $\boldsymbol{I_A}$ and it is annotated by $\boldsymbol{\widehat{I_A}}$'. In other words, $\boldsymbol{I_A \succ \widehat{I_A}'}$. Here, a problem is detected, and there exists many possible ways to inject a solution:

1) Entry-Correction: As shown in Fig.2, the input of the solution $\boldsymbol{S}$ can be injected at $\widehat{\boldsymbol{I_A}}$' taking into consideration that the system state at that point is equal or better than expected for the solution to be executed: $\boldsymbol{I_S \preccurlyeq \widehat{I_A}'}$. Whereas the output of $\boldsymbol{S}$ is injected before the Activity $\boldsymbol{A}$ taking into consideration $\boldsymbol{O_S \succcurlyeq \widehat{I_A}}$. In that case an ***entry-correction*** is made before entering an activity $\boldsymbol{A}$.

2) Bypass: (cf. Fig. 2) $\boldsymbol{S}$ can be injected at $\widehat{\boldsymbol{I_A}}'$ where $\boldsymbol{I_S \preccurlyeq \widehat{I_A}'}$ and it bypasses the activity $\boldsymbol{A}$ satisfying the following condition at the output of $\boldsymbol{S_1}$: $\boldsymbol{O_S \succcurlyeq \widehat{O_A}}$. In that case a *Bypass* is made. As an activity could be atomic or group of activities, it is essential to mention at this point that the *bypass* can be made to override one, two or as many blocks as required. But the output of the solution should always be compatible with the injection point on the main workflow. This type of correction is a novelty with respect to other approaches.

It is noteworthy to mention that there might be other ways to connect the output such as before or in the middle of the task. But these cases will not be considered in this paper as we consider only forward reparations.

*** When a task fails to execute:** This failure could be due to lack of resources (annotated by $\boldsymbol{\widehat{I_A} \prec R_A}$) or due to a specific constraint and the task will fail to produce any output. In this case the proposed repairs are:

1) Restore: (cf. Fig. 2) In that case, when a problem is identified and exception $\boldsymbol{E}$ is generated, the system will handle the exception and search for a suitable solution to restore task $\boldsymbol{A}$. The workflow will resume operation as soon as it gets an acknowledgment message $\overline{\boldsymbol{E}}$ that declares that the problem was solved.

2) Bypass: (cf. Fig. 2) The *Bypass* is used and Task $\boldsymbol{A}$ is replaced in the workflow by a solution $\boldsymbol{S}$ same as the *Bypass* in the previous paragraph when the problem occurred on the entry of the activity $\boldsymbol{A}$.

*** When a problem occurs at the output:** In this case, the output of Activity $\boldsymbol{A}$ is incompatible with the system state $\widehat{\boldsymbol{O_A}}$. ($\boldsymbol{O_A \prec \widehat{O_A}}$) and the only considered repair is:

Exit-Correction: (cf. Fig. 2) here similarly to what is previously discussed in the ***Entry-Correction***.

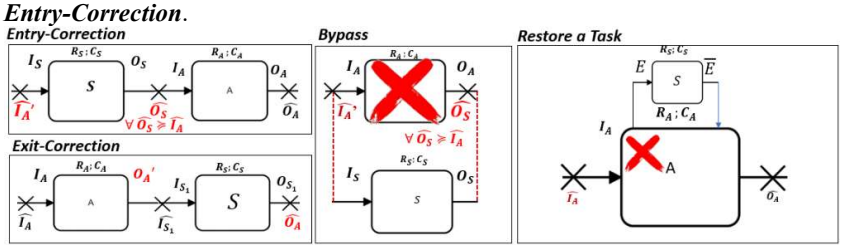


Fig. 2 Reparation Types

After applying a repair, the characteristics $(\boldsymbol{I, O, R, C})$ of the resulting model can be computed to be able to evaluate the impact of the reparation on the system. In fact, we already know that $(\boldsymbol{I_s, O_s}$ **and** $\boldsymbol{C_s})$ are satisfied otherwise we could not use

$\boldsymbol{S}$ in the reparation (cf 3.4). The modification then only applies to the resources used by the new model. Table 1 summarizes the impact of each reparation type on the usage of resources compared to the one used by the original model/system, in other words, before and after applying the reparation. The circled plus sign represents the aggregation of resources.

Table 1. Reparation patterns for Atomic Case.

Problem	Solution	Solution Pattern	Before	After
At the Input	Repair 1	***Entry-Correction***	R_A	$R_S \oplus R_A$
	Repair 2	***Bypass***	R_A	R_S
Task Failed	Repair 1	***Restore***	R_A	$R_{RestoreA}$
	Repair 2	***Bypass***	R_A	R_S
At the Output	Repair 1	***Exit-Correction***	R_A	$R_A \oplus R_S$

3.4 Selection of a sub-model from a library

In order to repair a problem, the system will search inside a library of sub-models for solutions that can satisfy the following criteria:

-Criterion 1: The input of the solution $\boldsymbol{I_S}$ is compatible with the system state:

$$\boldsymbol{I_s \preccurlyeq \widehat{I_A}}$$

-Criterion 2: The constraints $\boldsymbol{C_S}$ to execute a solution $\boldsymbol{S}$ are satisfied.

-Criterion 3: The resources $\boldsymbol{R_S}$ to execute a solution $\boldsymbol{S}$ are available

$$\boldsymbol{\preccurlyeq \widehat{I_A}}$$

-Criterion 4: The output of the solution $\boldsymbol{O_S}$ is compatible with the system state at the exit:

$\boldsymbol{O_S \succcurlyeq \widehat{O_A}}$.

It is essential to mention that solutions are regular activities that can be atomic or a group of tasks. Moreover, the library of solutions can always be extended by allowing users to collaborate, by always adding new solutions and fixes. The proposed approach consists of finding solutions in the collaborative library that are compatible with the actual system state where the system will not respond only to prescribed exceptions but also to the unexpected ones. Moreover, the reparation could imply or not the *Bypass* of one or many blocks respectively to each case by searching a returning point on the main model where $\boldsymbol{O_s}$ is compatible with the expected system state at that point. Fig.3 shows an illustration of the solution library.

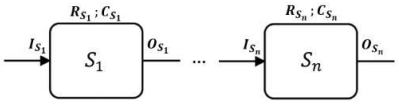


Fig. 3 Collaborative Solutions library

3.5 Bypass Exit-Point (Granularity)

As previously mentioned in (paragraph 3.3), the *Bypass* could affect one or more blocks on the main workflow. The granularity at this stage is defined by how large the replaced block can be. In the case where a group of tasks (annotated by $\boldsymbol{G}$) is bypassed, the respective group's properties are the aggregation properties of all the bypassed blocks. Consequently, the exit point of the solution should satisfy the constraints set $\boldsymbol{S_G}$, meet the required resources $\boldsymbol{R_G}$ and finally, match the output of the system at the point of injection. Finally, the Performance Indicator of the affected group will depend on the size of the replaced blocks and may be used to set the penalty caused by this replacement, or in other terms, it will contribute to computing the objective.

3.6 Evaluation and sorting

In order to evaluate the selected compatible set of solutions an objective set by the decision-maker is calculated in function of the Performance Indicators, the Overquality and the granularity of the proposed solutions. The objective function is annotated by:

$\boldsymbol{Obj(PI_1, \dots PI_n)}$ **where** $\boldsymbol{PI = \{Quality, Granularity, Cost, Interoperability \dots\}}$.

The Quality is defined as Overquality where the output $\boldsymbol{O_S}$ is much bigger than what is expected on the workflow as system state and annotated as $\boldsymbol{O_S \gg \hat{I}}$. Finally, the solutions will be ranked with respect to the objective function.

3.7 Decision Support

The decision can be configured by the decision-maker to be made manually or automatically by the system based on the *objective*. The objective is set by the decision maker, or a domain expert based on one or multiple performance indicator (s). In this last case, the solution ranked first with respect to the objective function will be selected. In the case of manual selection, all $\boldsymbol{PIs}$ as well as the *objective* are presented to the decision-maker for all the selected solutions to help him decide. The selected solution (local deviation) should repair the local problem and ensure the coherence of the global objective.

As an example, consider the set of solutions $\boldsymbol{S}$:

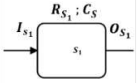
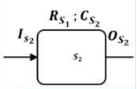
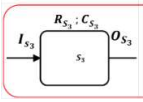
$$\boldsymbol{S = \{S_1(10€; 2ms), S_2(12€; 2ms), S_3(10€; 1ms)\}}.$$

The system will evaluate and rank the above solution set with respect to the objective function $\boldsymbol{Obj = \{Minimize\ cost\ first\ then\ minimize\ delay\}}$, where the considered indicators for this example are the time delay and cost: $\boldsymbol{PI = \{Delay, Cost\}}$**.** Consequently, the first sorting will be performed on the ***Less Cost***

indicator and then the second on the ***less delay*** indicator. In case the decision is made automatically, $\boldsymbol{S_3}$ is selected.

In case the user doesn't want to choose the reparation preselected automatically, a list with all possible reparations is presented, such as the list shown in Table 2. The decision-maker can then choose the one that best meets his needs according to his expertise in the domain. Thus, for instance, he could potentially make a different choice than the one expressed by the objective function based on his previous experience with a supplier (*e.g.*, an additional delay, a quality problem, an additional cost).

Table 2. Example of evaluation, ranking and decision.

Reparations	*PIs*	Cost Order	Time Order
	C 10 € T 2 *ms*	1	2
	C 12 € T 2 *ms*	2	2
	C 10 € T 1 *ms*	1	1

4 Conclusion and Perspectives

This paper introduced a new approach to repair unexpected situations on a workflow based on "simple" repairs but with complex verification and validity. The reparation will take advantage of a collaborative library to identify compatible solutions with respect to the state of the system. It can then gain benefits from external expertise on similar problems. Semantic verification and simulation will be used to ensure compatibility of the reparation with the system state. In literature, "reparations" are generally done either by a predicted deviation of the workflow or by raising an exception which is solved locally, and, in both cases, the original model remains unchanged. By opposite to these static approaches, this study introduces a dynamic approach by allowing to modify the model (inserting/deleting parts of the workflow). As the proposed approach is dealing with unexpected problems, then unpredicted reparations will be suggested. Compatible solutions are evaluated and sorted with respect to performance indicators. The decision-maker may express his needs by defining an objective function (based on these performance inductors) where the best solution may be automatically injected in the workflow. However, because of the unforeseen aspect of the problem, the method always allows the decision-maker to interact with the system and manually select

his preferred reparation. To achieve this, performance indicators and the sorting are available to assist the decision-maker in his choice.

As shown in this paper, one of the main problems is to identify where the reparation should return on the workflow. In future work, one of the major works would focus on this point by studying formal verification and simulation aspects, as well as the implications inferred from this such as uncertainty about the outcome. Discrete event simulation could be the key to study the dynamics and the information related to time delay and use of resources. In addition, basic reparations might be enhanced by introducing new features taking into account temporary reparation. Another aspect of the future work is to carry out complex reparation by combining simple repairs into one.

Acknowledgment: This research was partially funded by the BeePMN project from Program Hubert Curien CEDRE (France/Lebanon), grant number 4541TF

References

1. Rabah-Chaniour, S., Zacharewicz, G., Chapurlat, V.: Process Centered Digital Twin., GDR MACS (2022), Online Oral presentation, last access: 31 May 2022, https://action-jn.sciencesconf.org/resource/page/id/5
2. Ducq *, Y., Vallespir, B.: Definition and aggregation of a performance measurement system in three aeronautical workshops using the ECOGRAI method. Production Planning & Control. 16, 163–177 (2005).
3. Heguy, X., Zacharewicz, G., Ducq, Y., Tazi, S., Vallespir, B.: A Performance Measurement Extension for BPMN. In: Popplewell, K., Thoben, K.-D., Knothe, T., and Poler, R. (eds.) Enterprise Interoperability VIII. pp. 333–345. Springer International Publishing, Cham (2019).
4. Ougaabal, K., Zacharewicz, G., Ducq, Y., Tazi, S.: Visual Workflow Process Modeling and Simulation Approach Based on Non-Functional Properties of Resources. Applied Sciences. 10, (2020).
5. Kherbouche, O.M., Ahmad, A., Basson, H.: Using model checking to control the structural errors in BPMN models. In: IEEE 7th International Conference on Research Challenges in Information Science (RCIS). pp. 1–12 (2013).
6. Mallek, S., Daclin, N., Chapurlat, V.: The application of interoperability requirement specification and verification to collaborative processes in industry. Computers in Industry. 63, 643–658 (2012).
7. Borgida, A., Murata, T.: Tolerating Exceptions in Workflows: A Unified Framework for Data and Processes. In: Proceedings of the International Joint Conference on Work Activities Coordination and Collaboration. pp. 59–68. Association for Computing Machinery, New York, NY, USA (1999).
8. Svendsen, A.: Application Reconfiguration Based on Variability Transformations. Technical Report 2009-566 School of Computing, Queen's University Kingston, Ontario, Canada. 4 (2009).

9. Gao, H., Huang, W., Yang, X., Duan, Y., Yin, Y.: Toward service selection for workflow reconfiguration: An interface-based computing solution. Future Generation Computer Systems. 87, 298–311 (2018).
10. Rosa, M.L., Aalst, W.M.P.V.D., Dumas, M., Milani, F.P.: Business Process Variability Modeling: A Survey. ACM Comput. Surv. 50, (2017).
11. Gottschalk, F., van der Aalst, W.M.P., Jansen-Vullers, M.H.: Configurable Process Models — A Foundational Approach. In: Becker, J. and Delfmann, P. (eds.) Reference Modeling: Efficient Information Systems Design Through Reuse of Information Models. pp. 59–77. Physica-Verlag HD, Heidelberg (2007).
12. Becker, J., Delfmann, P., Knackstedt, R.: Adaptive Reference Modeling: Integrating Configurative and Generic Adaptation Techniques for Information Models. In: Becker, J. and Delfmann, P. (eds.) Reference Modeling: Efficient Information Systems Design Through Reuse of Information Models. pp. 27–58. Physica-Verlag HD, Heidelberg (2007).
13. Bayer, J., Kettemann, S., Muthig, D.: PESOA Process Family Engineering in Service-Oriented Applications BMBF-Project Principles of Software Product Lines and Process Variants. Presented at the (2004).
14. Hallerbach, A., Bauer, T., Reichert, M.: Issues in Modeling Process Variants with Provop. In: Ardagna, D., Mecella, M., and Yang, J. (eds.) Business Process Management Workshops. pp. 56–67. Springer Berlin Heidelberg, Berlin, Heidelberg (2009).
15. Possik, J., Zouggar-Amrani, A., Vallespir, B., Zacharewicz, G.: Lean techniques impact evaluation methodology based on a co-simulation framework for manufacturing systems. International Journal of Computer Integrated Manufacturing. 35, 91–111 (2022).
16. Russell, N., Aalst, W., Ter, A.: Exception handling patterns in process-aware information systems. (2006).
17. Adams, M., ter Hofstede, A.H.M., van der Aalst, W.M.P., Edmond, D.: Dynamic, Extensible and Context-Aware Exception Handling for Workflows. In: Meersman, R. and Tari, Z. (eds.) On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS. pp. 95–112. Springer Berlin Heidelberg, Berlin, Heidelberg (2007).
18. Jasinski, A., Qiao, Y., Fallon, E., Flynn, R.: A Workflow Management Framework for the Dynamic Generation of Workflows that is Independent of the Application Environment. In: 2021 IFIP/IEEE International Symposium on Integrated Network Management (IM). pp. 152–160 (2021).
19. Andree, K., Ihde, S., Pufahl, L.: Exception Handling in the Context of Fragment-Based Case Management. In: Nurcan, S., Reinhartz-Berger, I., Soffer, P., and Zdravkovic, J. (eds.) Enterprise, Business-Process and Information Systems Modeling. pp. 20–35. Springer International Publishing, Cham (2020).
20. About the Business Process Model and Notation Specification Version 2.0.2, https://www.omg.org/spec/BPMN/, last accessed 2022/05/27.
21. Rowell, D.: State-space representation of LTI systems. URL: http://web. mit. edu/2.14/www/Handouts/StateSpace. pdf. 1–18 (2002).