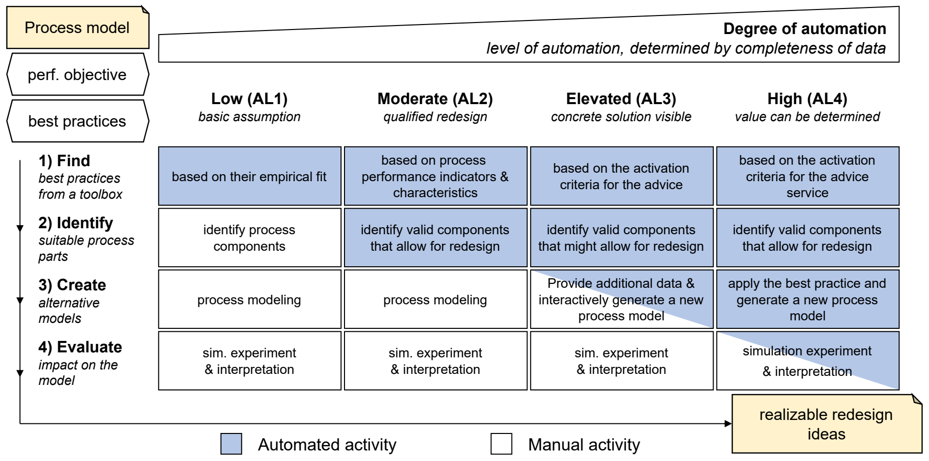
Assisted Business Process Redesign Modeler

Tobias Fehrer[a]

[a] *FIM Research Center, Universities of Augsburg and Bayreuth, Germany*

Abstract

For many organizations, the continuous optimization of their business processes has become a critical success factor. Several related methods exist that enable the step-by-step redesign of business processes. However, these methods are mainly performed manually and require both creativity and business process expertise, which is often hard to combine in practice. To enhance the quality and effectiveness of business process redesign, this paper presents a conceptualization of assisted business process redesign (aBPR). The aBPR concept guides users in improving business processes based on redesign patterns. Depending on the amount of data at hand, the aBPR concept classifies process redesign options into four types of recommendation that differ in their level of automation. Further, this paper proposes a reference architecture that provides operational support for the implementation of aBPR tools. The reference architecture has been instantiated as a prototype and evaluated regarding its design specification, applicability, and usefulness in artificial and naturalistic settings by performing an extensive real-world case study and interviewing experts from research and practice.



Process model
perf. objective
best practices
Degree of automation
level of automation, determined by completeness of data
Low (AL1)
basic assumption
Moderate (AL2)
qualified redesign
Elevated (AL3)
concrete solution visible
High (AL4)
value can be determined
1) Find
best practices from a toolbox
2) Identify
suitable process parts
3) Create
alternative models
4) Evaluate
impact on the model
based on their empirical fit
based on process performance indicators & characteristics
based on the activation criteria for the advice
based on the activation criteria for the advice service
identify process components
identify valid components that allow for redesign
identify valid components that might allow for redesign
identify valid components that allow for redesign
process modeling
process modeling
Provide additional data & interactively generate a new process model
apply the best practice and generate a new process model
sim. experiment & interpretation
sim. experiment & interpretation
sim. experiment & interpretation
simulation experiment & interpretation
realizable redesign ideas
Automated activity
Manual activity

Figure 1: Conceptualization of ABPR.

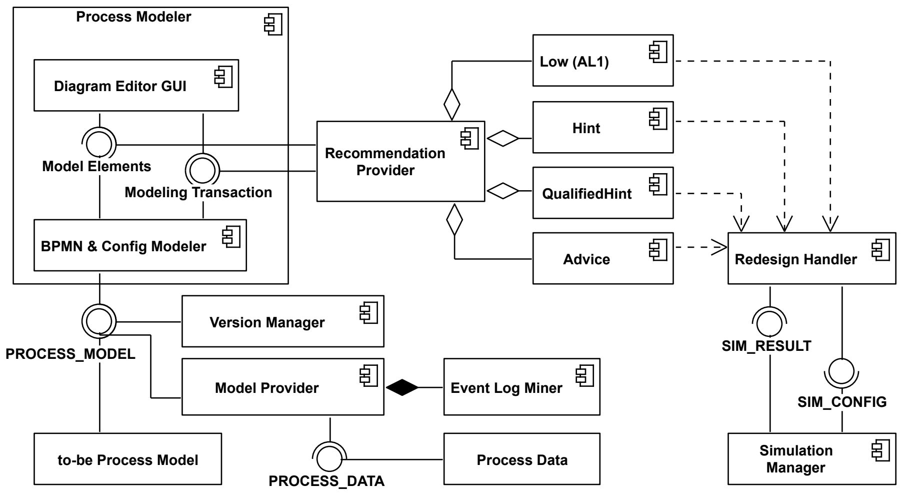


Figure 2: ABPR reference architecture.

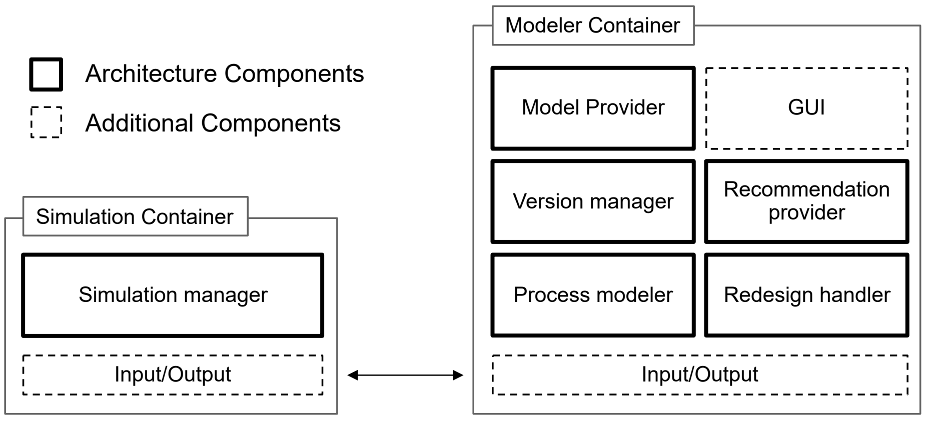


Figure 3: Prototype software architecture.

1. Introduction

2. Implementation

2.1. Prototype architecture design

To demonstrate the real-world applicability of our artifact [1], we instantiate the ABPR RA as a software prototype. Since our prototype represents a concrete instantiation of the RA presented in Figure 2, all components except for the optional event log miner are implemented. We further present the prototype with several specific redesign handlers with varying levels of automation and demonstrate the prototype with an exemplary process model.

We derive the prototype's specific architecture from the ABPR RA. The specific architecture incorporates design decisions that make the prototype an operational decision support system. We opt for an implementation that is built on top of web technologies.

The prototype is split into two container components, as can also be seen in Figure 3. The modeler container is an application that runs on the user's client. The

Email address: `tobias.fehrer@fim-rc.de` (Tobias Fehrer) simulation container is a cloud-based backend service. The decision to separate the simulation engine was made due to performance requirements: Since the component is to perform many simulation experiments in parallel, a scalable backend was chosen. The two containers communicate via web protocols. The modeler container is also built as a web application that can run in the user's browser as well as be integrated into the open-source Camunda Modeler[1]. For this, we heavily plug into the bpmn-js (v8.2.0)[2] JavaScript library. Since the prototype is intended to interactively assist the user in process redesign, the application is built around the graphical user interface (GUI) as its core. For programming we use the JavaScript libraries React (v16.8.6) and Fluent Ui (v7.160.2)[3]. In the backend we build the container environment with Express (v.4.17.1)[4]. The bundler is webpack (v4.30.0) with the Babel transcompiler (v7.4.3)[5].

2.2. Model provider

The meta models for the process data model are provided on top of the moddle[6] JavaScript library. The library uses meta-models to create, manipulate and validate process models and to (de-)serialize corresponding eXtensible Markup Language (XML) documents. We leverage the meta model library for the BPMN standard and its diagram notation (default XML namespace prefix: `bpmn`) that is provided in an open source repository. The meta-models for the simulation configuration (default XML namespace prefix: `bsim`), the simulation performance data (default XML namespace prefix: `bsimapi`), and annotations for recommendation-specific elements (default XML namespace prefix: `abpr`) are defined by the authors by adapting related works (e.g., [2]). **??** gives an example for the meta-model descrip-

[1]https://camunda.com
[2]https://github.com/bpmn-io/bpmn-js
[3]https://developer.microsoft.com/en-us/fluentui
[4]https://expressjs.com/
[5]https://webpack.js.org/
[6]https://github.com/bpmn-io/moddle

```
{
  "name": "task",
  "superClass": ["RootElement"],
  "properties": [
    { "name": "name", "type": "String", "isAttr": true },
    { "name": "bpmnElement", "isAttr": true, "isReference": true,
      "isVirtual": true,"type": "bpmn:Activity" },
    { "name": "resources", "type": "resources" },
    { "name": "duration", "type": "duration" },
    { "name": "setUpDuration", "type": "setUpDuration" },
    { "name": "boundaryEvents", "type": "boundaryEvents" }
  ]
}
```

Listing 1: Type descriptor a task element.

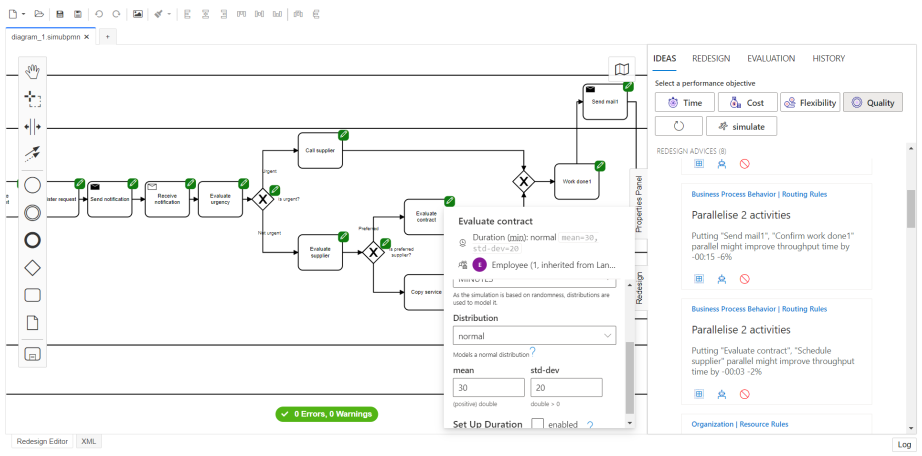
tion for the `bsim:task` element. As can be seen, an instance of this type refers to an `bpmn:Activity` element and additionally describes the resource assignment, activity duration, an optional set-up duration, and optional boundary events attached to the activity that are relevant for both simulation and redesign recommendation generation.

Section 2.2 shows an exemplary task element in serialized XML form. In total, we define 80 type descriptors to support an extensive set of process elements in a library collection[7]. In the prototype, we use the factory programming pattern to provide a standardized interface for instantiating new model elements. When parsing a standard BPMN diagram, resources, and their costs, activity durations, gateway split probabilities, resource timetables, and further properties are initialized with a default configuration to ensure valid models and a convenient experience for the user.

[7]https://github.com/dtdi/bsim-moddle

```xml
<bsim:Task id="Activity_0v155hv" name="Receive Request">
  <bsim:resources>
    <bsim:resource id="Employee" amount="1" />
  </bsim:resources>
  <bsim:duration timeUnit="MINUTES">
    <bsim:normalDistribution>
      <bsim:mean>15</bsim:mean>
      <bsim:standardDeviation>4</bsim:standardDeviation>
    </bsim:normalDistribution>
  </bsim:duration>
  <bsim:setUpDuration />
  <bsim:boundaryEvents>
    <bsim:boundaryEvent id="Event_1w29ro5">
      <bsim:arrivalRate timeUnit="MINUTES">
        <bsim:normalDistribution>
          <bsim:mean>20</bsim:mean>
          <bsim:standardDeviation>5</bsim:standardDeviation>
        </bsim:normalDistribution>
      </bsim:arrivalRate>
      <bsim:eventProbability>1</bsim:eventProbability>
    </bsim:boundaryEvent>
  </bsim:boundaryEvents>
</bsim:Task>
```

Listing 2: Serialized $bsim : Task$ element with resource assignment, duration, and an attached boundary event.



Evaluate contract
Duration (min): normal `mean=30, std-dev=20`
Employee (1, inherited from Lan...
As the simulation is based on randomness, distributions are used to model it.
Distribution: normal
Models a normal distribution
mean: 30 ((positive) double)
std-dev: 20 (double > 0)
Set Up Duration: enabled

IDEAS | REDESIGN | EVALUATION | HISTORY
Select a performance objective
Time | Cost | Flexibility | Quality
simulate
REDESIGN ADVICES (8)

Business Process Behavior | Routing Rules
Parallelise 2 activities
Putting "Send mail1", "Confirm work done1" parallel might improve throughput time by -00:15 -6%

Business Process Behavior | Routing Rules
Parallelise 2 activities
Putting "Evaluate contract", "Schedule supplier" parallel might improve throughput time by -00:03 -2%

Organization | Resource Rules

Figure 4: Extended modeling interface for the simulation properties of an activity element.

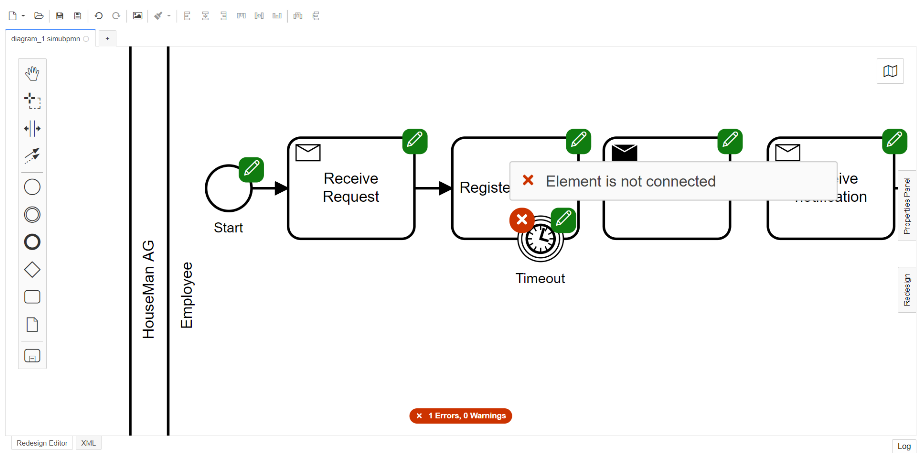
2.3. *Process modeler*

We extend the bpmn-js library to implement the process modeler component. Our extensions ensure that user interactions are valid and provide a common user interface for the model extensions and redesign settings (see Figure 4). Once the user creates, modifies, or deletes a shape in the modeler, the underlying process data model is adjusted accordingly.

The available shapes are restricted to a set that is supported by the redesign handler and that can be mapped to the process data model. A model linter identifies problems in the process. As visualized in Figure 5, the linter detects misconfigurations or missing properties and provides the user with visual feedback to fix the model. Each of our meta-model extensions is safeguarded by custom linting rules.

2.4. *Simulation manager*

The simulation experiments are performed in a cloud-based backend service, the simulation container. We build on the business process simulator Scylla [2], which we migrate from a desktop application to a remotely invocable service worker. To execute a simulation experiment, the process model, a simulation configuration, and global simulation properties are submitted to the service in XML format. The



Element is not connected

1 Errors, 0 Warnings

Figure 5: Visual feedback during process modeling indicates configuration problems.

simulation engine will then return information on cost, time, and utilization of activities and resources in XML as well. Web protocols serve as a layer between components.

2.5. Redesign handler

The redesign handlers plug into the bpmnjs library and are implemented from scratch. Each redesign handler implements a specific type of recommendation from a generic interface that prescribes its lifecycle through several methods that are executed independent from the level of automation: (1) `Recommendation#execute()`, (2) `Handler#initialize()`, (3) `Handler#evaluate()`, (4) `Handler#commit()`, (5) `Handler#finalize()`, (6) `Handler#revert()`, and (7) `Handler#stash()`. The different states of a redesign handler are represented in the state machine diagram in Figure 6.

The `execute()` method of class Recommendation, is the environment that controls the operation of a redesign handler. By traversing the process model, the method identifies suitable process parts for applying the redesign best practice. For the recommendation type advice, this method will already trigger the `initialize()` and the `evaluate()` method, to automatically perform and evalu-

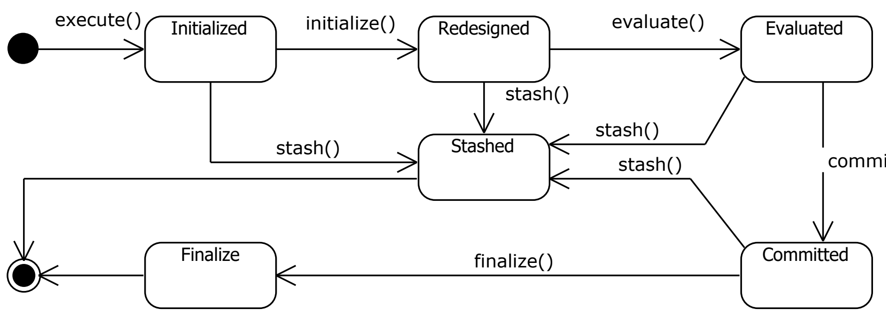


Figure 6: State machine representation of a redesign handler.

ate a process redesign in the background. The results of this evaluation can then be used to return the impact of this redesign to the recommendation provider. In the remaining types of recommendation, where automatic redesign cannot be performed without user interaction, no further lifecycle is executed without the user and the recommendation is returned to the recommendation provider. In `initialize()`, the redesigned process model is generated. Also, the status of the process model is saved and all individual redesign commands that lead to the new process model are tracked in a changeset. The redesign can be executed either via user interaction with the process modeler or programmatically via implemented redesign algorithms. The method `evaluate()`, initiates a simulation experiment in the simulation manager. The results of this simulation run are combined with the simulation result of the base model and processed for comparison. This means, the deviance between the actual model and the redesign option is additionally calculated in percentage change and absolute difference in the corresponding unit. We provide several process performance aspects that are broken down to various performance indicators to the user as presented in Table 1.

By executing `commit()`, the user accepts the redesigned process model. The user may provide an optional commit message to annotate the changeset. In `finalize()`, the user finally approves the redesign and finishes the current redesign. The handler's lifecycle is completed, and a new iteration can begin. Dur-

Table 1: Simulation data returned from available in the evaluation method

Property	Dimension	Unit
Global Time	Effective Time	Time (d hh:mm:ss)
	Flow Time	Time (d hh:mm:ss)
	Off Timetable	Time (d hh:mm:ss)
	Waiting Time	Time (d hh:mm:ss)
Global Cost	Process Cost	Monetary unit (€)
Resources (per resource)	Cost	Monetary unit (€)
	Workload	Percentage
Activities (per activity)	Duration	Time (d hh:mm:ss)
	Cost	Monetary unit (€)
	Instances	Integer
	Resource Idle Time	Time (d hh:mm:ss)
	Activity Waiting Time	Time (d hh:mm:ss)

ing the lifecycle, the user can choose to cancel the redesign and discard all changes made to the process model by executing `stash()`. This as well puts the redesign in a final state and a new iteration can be invoked.

2.6. Recommendation provider

The recommendation provider is implemented as an additional GUI component to the process modeler. It allows the selection of one performance objective for the generation of useful recommendations and lists suitable recommendations in a list. The list is split into two groups. The upper group contains only (guided) advices, whereas the remaining recommendations (i.e., hints and qualified hints), are kept in a second group below. The recommendation provider triggers the generation of recommendations on certain events, for example, when starting the application, loading a new process model, or upon finalizing a redesign. The different recommendations can be identified by their type and the best practice's unique name (e.g., DU-10 for the best practice "extra resources"). Several recommendations might be generated, also for different types. Each recommendation may choose how recommendations from the same pattern may be treated and choose to remove them. For example, a "DU-10.TYPE_ADVICE" recommendation overrides "DU-10" recommendations of type TYPE_IDEA and TYPE_HINT.

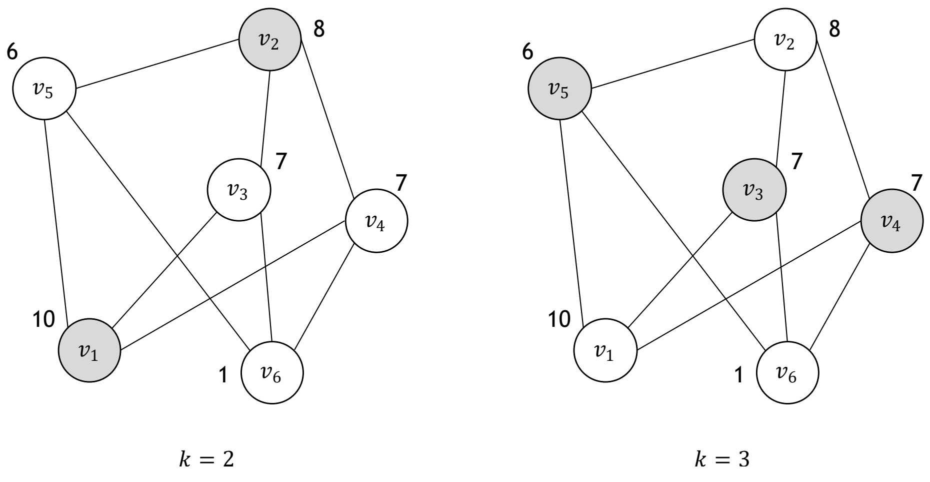


Figure 7: Exemplary diversity graph with several nodes v_i, their scores, similarity edges, and the optimal solutions for $k = 2$ and $k = 3$ from [3].

For recommendation diversification, an $A*$ algorithm for diversified top-k results [3] is utilized. Based on both, the contribution $score(v_i)$, and the similarity within the recommendations, $sim(v_i, v_j)$, this algorithm generates a set $D(S)$ of user-adjustable length k with $|D(S)| \leq k$ from a set of recommendations $S = v_1, v_2, \ldots$. The approach aims to reduce redundancy while maintaining the quality in the recommendations. For searching $D(S)$, Qin et al. [3] propose an undirected diversity graph $G(S) = (V, E)$ such that for any result $v \in R$, there is a corresponding node $v \in V$, and for any two results $v_i \in S$ and $v_j \in R$, there is an edge $(v_i, v_j) \in E \iff v_i \approx v_j$. The authors state, that $v_i \approx v_j$ is given, if the similarity between two recommendations exceeds a certain threshold $\tau : sim(v_i, v_j) > \tau$. An exemplary diversity graph is shown in Figure 7. The illustration also shows how sets of different length are generated.

We implemented the data structure and the solution algorithm as a generic standalone JavaScript library for the use in our prototype. A custom scoring function estimates the $score(v_i)$ for each recommendation. The score is calculated in line with the configured performance objective. If a recommendation cannot determine a concrete impact on the performance objective, we consider the heuristics' general performance indication [4] that is mapped to a numeric range $[-1;1]$ as the best possible score.

To estimate whether $v_i \approx v_j$, a similarity function $sim(v_i, v_j)$ considers recommendation properties. The similarity is calculated as a measure that considers information specific to the generic best practice (e.g., process aspect, best practice name) and the concrete instantiation of the recommendation (e.g., the similarity of affected elements). The similarity threshold τ has been manually tuned and set to 0.2 over several iterations.

From the two generated lists, the user can inspect the different recommendations which are displayed as card GUI elements (see box 3 in Figure 10). The GUI cards provide the process aspect, the heuristic category, its name, a short description, and optionally the expected redesign impact and affected process elements. On the card, the user has the option to `initialize()` the redesign handler, to navigate to the corresponding process part, to watch an explanation video (implemented for some heuristics only), or to hide the recommendation permanently if it does not suit the redesign objective.

2.7. Version manager

The different model versions are captured in a stack of applied redesigns. Several changes that constitute a new process design are summarized as a changeset that the user can annotate with a description. Individual changes, as well as whole changesets, can be reverted to allow for flexible modeling. Using the changesets, several process alternatives can be tracked and documented for iterative improvement.

2.8. Implemented redesigns

In our prototype, we support all patterns from Reijers and Limam Mansar [4] in varying levels of automation. An overview of the different patterns implemented in

Table 2: Supported redesign bet practices grouped by the type of recommendation.

Type	Best practice
Idea	Control relocation (DU-01), Contact reduction (DU-02), Integration (DU-03), Case types (DU-04), Activity elimination (DU-05), Case-based work (DU-06), Triage (DU-07), Activity composition (DU-08), Re-sequencing (DU-09), Parallelism (DU-10), Knock-out (DU-11), Exception (DU-12), Case assignment (DU-13), Flexible assignment (DU-14), Centralization (DU-15), Split responsibilities (DU-16), Customer teams (DU-17), Numerical involvement (DU-18), Case manager (DU-19), Extra resources (DU-20), Specialist-generalist (DU-21), Empower (DU-22), Control addition (DU-23), Buffering (DU-24), Activity automation (DU-25), Integral technology (DU-26), Trusted Party (DU-27), Outsourcing (DU-28), Interfacing (DU-29)
Hint	Parallelism (DU-10)
Guided Advice	Triage (DU-07), Activity automation (DU-25)
Advice	Parallelism (DU-10), Extra resources (DU-20)

the prototype is given in Table 2. In the following, we describe the implementation of two exemplary patterns.

2.8.1. Parallelism Advice

The parallelism advice (DU-10.ADVICE) suggests putting sequential tasks in parallel [4]. The potential effect of this pattern is that the throughput time might be reduced [4]. This type of recommendation is generated fully automated. Therefore, all the steps, find, identify, redesign, and evaluate are executed in the background and then presented to the user, who can then decide to apply the recommendation. In the following, we describe the individual steps that are executed in sequence.

Find: This redesign best practice can be promoted if the goal of the redesign is to improve on the time dimension. In this case, the $execute()$ method is executed to trigger the subsequent steps.

Identify: We draw from the decision guidelines for finding suitable activities in existing work [5] and adjust them for our implementation. That is, putting activities in parallel can only be done and have a considerable effect if the following conditions are satisfied:

- The activities are in sequence.
- The activities have no data dependency upon one another.

- The duration times of the activities are of the same order of magnitude.
- Resources from different roles execute the activities (or, there is more than one resource available with that role).
- There is no overloading of any role because of putting activities in parallel, i.e., the resulting utilization rates are acceptable.

In a first step, we, therefore, identify activity sequences in the process. A sequence is a set of `bpmn:Task` elements in a straight sequence. To identify these sequences, we first retrieve potential start elements of sequences. A start element is any element whose direct predecessor matches any of these conditions: (1) Type is `bpmn:StartEvent`, (2) type is `bpmn:IntermediateCatchEvent`, (3) `length(bpmn:outgoing) > 1`, or (4) `length(bpmn:incoming) > 1`. The set of elements that qualifies as start elements is used to retrieve sequences. A recursive procedure adds any successor to the sequence until the next element matches any of these conditions: (1) Type is not `bpmn:Task` (2) `length(bpmn:outgoing) > 1`, or (3) `length(bpmn:incoming) > 1`. The data and the resource perspective are used to decide which activities in a sequence can be executed in parallel. Figure 8 shows how data dependencies influence the identification of candidates. Since the output data elements of activity A are the input data elements of activities B and C, activity B and activity C must be placed after activity A. Activity B does not create output data that is input data for activity C and vice versa, so activity B and activity C can be executed in parallel. Further, activities B and C are executed by different roles. In this step, we also filter out any activities that have been marked by the user as non-parallelizable together with other activities. This information is added to the model as an `abpr:property` in the `bpmn:ExtensionElements` container of the activity, if a user discards a parallelization recommendation. Each set of activities that fulfills the above conditions is handed to the redesign step as an individual recommendation candidate.

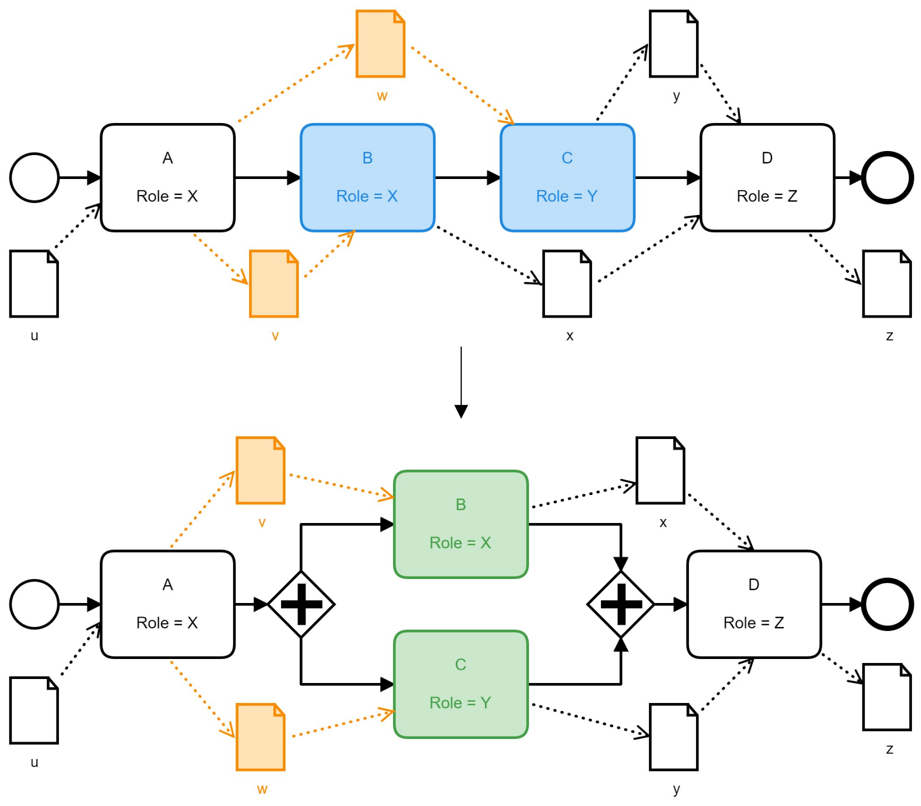


Figure 8: General application of the parallelism best practice from Netjes et al. [5, p. 18]

Redesign: The redesign is created by (1) adding parallel gateways before and after the activities, and (2) recreating the connections between the activities. The result can be seen in Figure 8.

Evaluate: The result of the simulation in comparison with the base simulation should show, that (1) the overall throughput time did decrease, and (2) the resource utilization is at an acceptable level. If both conditions apply, then the recommendation will be passed to the recommendation provider.

2.8.2. Triage Guided Advice

The triage best practice (DU-07.GUIDED_ADVICE) suggests to "consider the division of a general task into two or more alternative tasks" or "consider the inte-

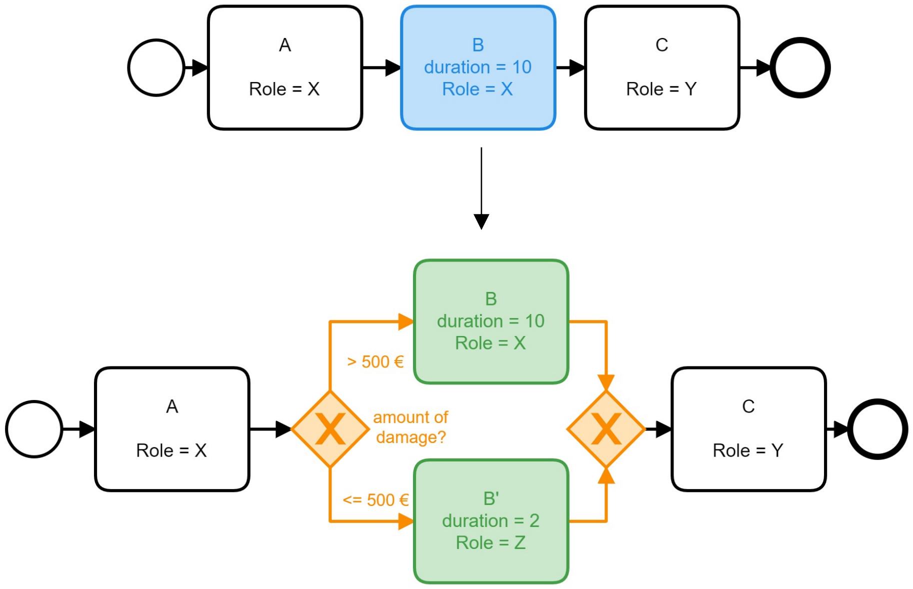


Figure 9: General application of the triage best practice.

gration of two or more alternative tasks into one general task" [4, p. 297]. In our implementation, we consider the first case.

Find: This redesign best practice can be promoted if the goal of the redesign is to improve on the quality dimension. As no further recommendation can be given without additional user input, a recommendation card is created at this step that helps to identify possible candidates.

Identify: We draw from the decision guidelines for finding suitable activities in existing work [4] and adjust them for our implementation. That is, dividing a general activity into alternative activities can only have a considerable effect if the following conditions are satisfied:

- A split criterion is defined to distinguish different cases or resources.
- The information to make the split decision is available before executing the

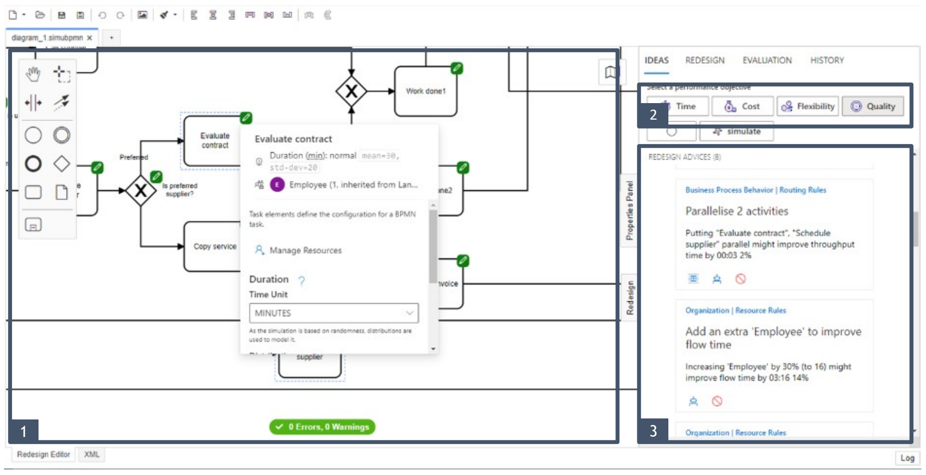
activity.

- Different treatments can be applied to different types of cases.

As the examination of these conditions relies on information that is not available in the process model, the implementation relies on expert input. The user marks activities that qualify for the application of the triage pattern. Therefore, a wizard is instantiated in the `initialize()` method. The wizard provides a list of all activities in the process with their description and resource assignments together with selectable options that suggest that the activity (1) is a potential candidate for the triage pattern, (2) does not qualify as a candidate for the triage pattern, or (3) is not considered (default). Upon selecting an activity as a potential candidate, the user is asked to provide the split criterion for routing the cases. The user choice and input are added to the model as `abpr:property` elements in the activity's `bpmn:ExtensionElements` property container.

Redesign: In the redesign step, further user input is required to model the specific triage best practice. Therefore, a second wizard guides the user through the redesign. In addition to the splitting criterion, the branching probabilities must be specified for each branch so that they add up to 100%. Further, for each alternative, the activity is modeled by specifying a name, assigning new resources, and configuring the execution durations. The process diagram is redesigned by (1) inserting exclusive gateways before and after the activity under consideration, (2) creating the alternative activities, and (3) recreating the connections between the activities with their branching probabilities. The result of the applied redesign can be seen in Figure 9.

Evaluate: Quality cannot directly be interpreted from the simulation results. In addition to this information, the expert user, therefore, is asked to interpret the quality of the redesign himself.



Evaluate contract

Duration (min): normal mean=30, std-dev=20

E Employee (1. inherited from Lan...

Task elements define the configuration for a BPMN task.

Manage Resources

Duration

Time Unit

MINUTES

As the simulation is based on randomness, distributions are used to model it.

✓ 0 Errors, 0 Warnings

REDESIGN ADVICES (8)

Business Process Behavior | Routing Rules

Parallelise 2 activities

Putting "Evaluate contract", "Schedule supplier" parallel might improve throughput time by 00:03 2%

Organization | Resource Rules

Add an extra 'Employee' to improve flow time

Increasing 'Employee' by 30% (to 16) might improve flow time by 03:16 14%

Organization | Resource Rules

Figure 10: Software prototype - general overview with GUI elements (1) diagram editor, (2) goal selection, and (3) list of recommendations.

3. Prototype demonstration

The prototype starts from an empty canvas or a preexisting BPMN process model imported from other applications. The prototype allows the user to annotate and edit the process model and generates redesign recommendations based on the model. The user can then apply these recommendations and evaluate their impact with simulation experiments. This procedure is iterated until the user is pleased. Finally, the improved process model can be exported as a BPMN file including all additional annotations and design decisions for later revisions. Figure 10 illustrates the GUI. For a prototypical application, we redesign an artificial service request process [5] that has been designed within the prototype from scratch. We further created a demo video[8] that illustrates how the prototype assists and semi-automates the redesign of this artificial process model.

We seek to improve the overall throughput time. Upon starting the application, time is already selected as the performance objective (see box 2 in Figure 10). Au-

[8]https://www.youtube.com/watch?v=VqrHj7RaFXQ

tomatically, a simulation experiment for the base scenario is executed and recommendations are generated. The user can click on the tab EVALUATION to examine the simulation results. As some recommendations build on simulation experiments themselves, the list of recommendations in the tab IDEAS is progressively enlarged within a few seconds. As we seek to improve on the time dimension, the top recommendations of type advice suggest parallelizing two activities. A click on the "find in model" button (the dashed grid) on the recommendation card reveals the two activities "Send mail1" and "Confirm work done1". The simulation suggests that parallelizing these two activities will decrease throughput time by 6%. Since the process model incorporates data dependencies and no dependencies are modeled between these two activities, we can apply this redesign. Automatically, the process model is remodeled, and the tab is activated where all model changes are listed.

A click on COMMIT will reveal the EVALUATION tab. Analyzing the results and considering the domain knowledge, we can conclude that the redesign is a valid way to improve the process. Therefore, we finalize the redesign and enter a commit message into the corresponding field, to describe our change. After completion, we end up in the IDEAS tab, where again, recommendations are generated under consideration of the changed process model.

Another recommendation suggests identifying activities where the triage pattern may be applied. We seek to apply the triage pattern in a way that automates cases that match certain conditions. That is, a human specialist may focus only on complicated cases. A click on the recommendation card reveals a wizard, that lists all activities. We select the "register request" activity as a candidate for the triage pattern. A text box is revealed, that captures the split criteria for the cases. The IDEAS tab now presents a specific recommendation that suggests applying the triage pattern to the previously selected activity. Selecting the recommendation "Apply triage to register request" therefore reveals wizard that supports remodeling the process.

From own judgment, we expect 80% of the cases to be submitted in a form that can be handled via automation in a short duration and reduced cost. The remaining 20% remain with the standard resource and duration configuration. Closing the wizard reveals the re-modeled process and a click on COMMIT executes the simulation experiment. The simulation reveals, that, as expected, the throughput time decreases. At the same time, the cost will be reduced, as certain cases are handled by an automation resource. The redesign can be considered useful and confirmed with a click on FINALIZE. Finally, the redesigned process model can be exported as a BPMN file for later use.

References

[1] S. T. March, G. F. Smith, Design and natural science research on information technology, Decis. Support Syst. 15 (1995) 251–266. doi:10.1016/0167-9236(94)00041-2.

[2] L. Pufahl, T. Y. Wong, M. Weske, Design of an extensible bpmn process simulator, in: Business Process Management Workshops, Springer, 2018, pp. 782–795. doi:10.1007/978-3-319-74030-0_62.

[3] L. Qin, J. X. Yu, L. Chang, Diversifying top-k results, in: Proceedings of the VLDB Endowment, 2012, pp. 1124–1135. doi:10.14778/2350229.2350233.

[4] H. A. Reijers, S. Limam Mansar, Best practices in business process redesign: an overview and qualitative evaluation of successful redesign heuristics, Omega 33 (2005) 283–306. doi:10.1016/j.omega.2004.04.012.

[5] M. Netjes, H. A. Reijers, W. M. P. van der Aalst, The PrICE tool kit: Tool support for process improvement, in: Business Process Management (Demos), Springer, 2010, pp. 58–63.