# An Idea On Infinite Horizon Decision Support For Rule-based Process Models

Michaela Baumann*, Michael Heinrich Baumann†, and Stefan Jablonski*
*Institute for Computer Science
†Institute for Mathematics
University of Bayreuth, Germany
Email: {michaela.baumann,michael.baumann,stefan.jablonski}@uni-bayreuth.de

***Abstract*—In recent years, process models tend to turn away from common procedural models to more flexible, rule-based models. The models are characterized by the fact that in each execution step users usually have to decide between several rule-consistent tasks to perform next. Precise execution paths are not given, which is why adequate execution support needs to be provided. Simulation is one means to facilitate the users' decisions. In this context, we suggest an execution simulation tool with an infinite horizon, i.e., in each (simulated) step, users are informed about the tasks that in any case still need to be done to properly finish one process instance, and about tasks that may no longer be executed. The forecasts consider an actual or a simulated history of the process instance and the rules given by the model.**

***Keywords–Process execution; Rule-based process models; Process decision support;***

## I. INTRODUCTION

In many fields of economy, industry, and research, process models are used for supporting the execution of operating processes, for designing work steps, for documentation purposes, etc. Usually, these process models are a sort of procedural process models, where the execution order of the process steps is prescribed through the control flow. Other execution orders than the prescribed ones are not provided. This is why computational offloading ("the extent to which differential external representations reduce the amount of cognitive effort required to solve informationally equivalent problems" [1]) is quite well achieved in procedural process models. For rule-based process models, this is not the case [2], as they take a different modeling and representation approach. They are typically used when procedural process models are too restrictive or get too complicated when complex facts shall be displayed. The approach of rule-based process models is to provide a set of tasks, firstly without stating any execution order, and then to restrict all possible execution orders by adding rules or constraints that should be met during the execution. An example for such a rule could be: "If task $A$ has been executed, afterwards task $C$ needs to be eventually executed, too". Thus, especially for rule-based process models, guidance for the user through the process is necessary, as the execution sequences leading to a proper process completion are not easy to see [3].

In this paper, we do not want to answer the question of which tasks may be executed in the next step, with a certain process history underlying. This has been done in other work, e.g., in [4] for ConDec models via automata, and is not part of the work at hand. Tasks for the next step have to be chosen in a way that every resulting process history is model conform and that dead ends are avoided. Furthermore, we need process models that do not contain conflicting constraints [4].

For run-time support, recommendations for effective execution [5] can be given. However, these recommendations are usually based on past experiences and need a specific goal, i.e., a rating of experiences in terms of desirability [6], as input. Parts of the executable tasks are hidden from the executing agent, i.e., a preselection has occurred. The decision support we head for is somehow different, as we do not intend to give recommendations based on a specific goal (as input into the system) but to provide the agent an overview over *the impact of each of his decisions*. He can then decide, according to the overview and a goal (which is only in his mind), which step to execute next. The system and the model do not need to be changed, which may cause history-based violations when done at run-time [6]. The questions that shall be answered by the support are the following: "Which tasks still need to be executed during the process instance?", "Which tasks may/can still be executed eventually during the process instance?", "What changes apply to the answers of the two preceding questions if one (or more) certain task is executed next?" As one can see, there is no limit of steps till the end of an instance for answering these questions, which is why we talk of *infinite horizon* in this context. A use case for this approach could be the following example situation: An employee has noticed that his colleague is overloaded with work, and thus he wants to finish the process without involving this colleague, i.e., avoid certain tasks, if possible.

The work proceeds as follows: Section II proposes the infinite horizon decision support with help of examples, Section III concludes with some features of the approach, remaining questions that still need to be answered, and suggestions for future work.

## II. IDEA: INFINITE HORIZON DECISION SUPPORT

We want to present our approach with a short example. Therefore, we consider four rules: the $existence$ rule, the $response$ rule, the $precedence$ rule, and the $chainResponse$ rule. They are defined as follows:

i) $existence(A, m, n)$: Task $A$ must at least be executed $m$ times and may at most be executed $n$ times ($m \leq n$)
ii) $response(A, B)$: If task $A$ appears in the process instance, then task $B$ has to appear after $A$, too
iii) $precedence(A, B)$: Task $B$ can only be executed if task $A$ has already been executed, i.e., already appears in the process history
iv) $chainResponse(A, B)$: Every execution of task $A$ has to be directly followed by $B$

 ISBN: 978-1-61208-432-9 

$\forall A \in \mathcal{A}$: $man(A) \leftarrow 0$; $opt(A) \leftarrow \infty$;

Figure 1: Initialization of mandatory and optional values

$\forall\ existence(A, m, n) \in \mathcal{R}$:
If $start$: $man(A) \leftarrow m$; If $start$: $opt(A) \leftarrow n$;

Figure 2: Update rules for process rule $existence$

For every task in the process model two states are recorded: *mandatory* and *optional*. The initialization of these states is conform to the rule-based approach. Let $\mathcal{A}$ denote the set of all tasks and $\mathcal{R}$ the set of all rules. At first, without considering any rules, no task must be executed ($\forall A \in \mathcal{A}: \ man(A) = 0$) but may be executed arbitrarily often ($\forall A: \ opt(A) = \infty$). The process history is stored in variable $h$. At the beginning, the history is empty: $h = \odot$. The task executed in the previous step is given by $\ell(h) \in \mathcal{A} \cup \{\texttt{NA}\}$.

After initialization of the status values for all tasks (Figure 1), the values are sequentially restricted according to the update rules (Figures 2–5), where function $start$ denotes the beginning of a process instance and $exec(\cdot)$ the execution of a task. After each task execution, both status values of the corresponding task are reduced by 1, if possible, before considering the update rules and adjusting the status values according to them. The list of all rules is processed sequentially, as many times, until in one run nothing more changes.

The update rules can be divided into three different kinds of rules. One type are the start and execution rules (If $start$, If $exec(\cdot)$). The start rules only need to be processed once after starting the process and can be skipped for the rest of the process after the first task execution. The execution rules need to be processed once after each task execution and can be skipped at the beginning. The second type are the indirect status update rules (all other rules in the example Figures 2–5 except for the last one in Figure 5), triggered through chain reactions caused by start and execution rules. Rules of the third type need to hold permanently and have no special trigger constraint, like the last rule in Figure 5 or the rule $opt(A) \geq man(A)$. The reason for the last update rule (resulting from $chainResponse$) in Figure 5 is: As after $A$, task $B$ must always follow directly, then $B$ needs to be done at least as many times as $A$ (plus 1, if $A$ was the most recently task). If $B$ needs to be done more often anyway, then nothing changes.

A possible prototype could look like the design draft in Figure 6, where two situations are shown. After having

$\forall\ response(A, B) \in \mathcal{R}$:
If $exec(A)$: $man(B) \leftarrow \max\{man(B), 1\}$;
If $man(A) > 0$: $man(B) \leftarrow \max\{man(B), 1\}$;
If $opt(B) == 0$: $opt(A) \leftarrow 0$;

Figure 3: Update rules for process rule $response$

$\forall\ precedence(A, B) \in \mathcal{R}$:
If $opt(A) == 0 \wedge A \notin h$: $opt(B) \leftarrow 0$;
If $man(B) > 0 \wedge A \notin h$: $man(A) \leftarrow \min\{man(A), 1\}$;

Figure 4: Update rules for process rule $precedence$

$\forall\ chainResponse(A, B) \in \mathcal{R}$ :
If $exec(A)$: $man(B) \leftarrow \max\{man(B), 1\}$;
If $opt(B) \neq \infty$:
  $opt(A) \leftarrow \min\{opt(B), opt(A) - \mathbb{1}_{\ell(h)==A}\}$;
$man(B) \leftarrow \max\{man(B), man(A) + \mathbb{1}_{\ell(h)==A}\}$;

Figure 5: Update rules for process rule $chainResponse$

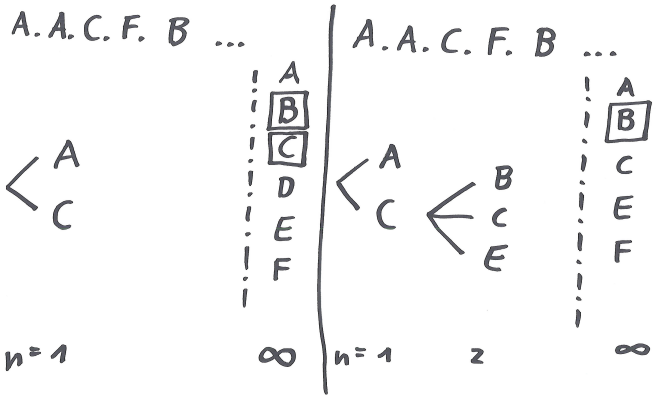


Figure 6: Prototypical design for an infinite horizon decision support tool

executed tasks A, A, C, F, and B, the tasks that are executable next in the first situation are tasks A and C. Note, that the update rules do not derive these next-executable tasks (that one with horizon step $n = 1$). The update rules rather determine the column on the right ($n = \infty$), which says that there exist possible execution paths for each task A to F, and that tasks B and C need to be executed in all of these paths to successfully finish the process execution. In the second situation, the history is still the same, but it is simulated how the infinite horizon changes if task C would be executed next (if the history was A.A.C.F.B.C). Now, task D may no longer be executed, no matter which task is chosen next (B, C, or E), and the status of C changes from mandatory to optional, so, $man(C) = 0$ and $opt(C) > 0$. This information, especially the infinite horizon after the simulation step (simulated execution of $C$), may help the agent to decide what to do next. The decision support can be expanded by using past execution histories to provide the agent information about average execution time for each step or about success rate of certain histories [5].

If the underlying process model is changed, then the set of update rules needs to be changed, too. The update rules corresponding to removed process rules, or even removed tasks, have to be eliminated, whereas new update rules, caused by added process rules, are included. For running process instances, there may occur two situations [7]: The current history is conform to the new set of process rules, then the new status values can be achieved by simulating their evolution according to the new set of update rules. If a so-called "history violation" [7] occurs, then we refer to [7] for handling the problem.

## III. Features, Remaining Questions, and Future Work

The idea paper suggests a possibility for decision support for declarative process models. This decision support applies to an infinitely long forecast horizon. It makes use of the

process rules and their implications to the states (mandatory and optional) of the tasks they refer to. At the moment, the focus lies only on rules concerning control-flow. It should be investigated if and how an extension to other process perspectives, like data and agents, is possible. Furthermore, instead of regarding tasks as single points in time, i.e., only their final execution is registered, it would be beneficial to split one task into different events, at least start and end. Nesting of tasks (subprocesses) could also be analyzed.

Repeatability and optionality of tasks [8] may be read off the status values, as well as dead activities when regarding the status values with empty history $h = \odot$. If one task $A$ has $opt(A) = 0$ at the beginning after the first evaluation round of the update rules, it can never be executed. The question arises if it is possible to modify update rules so that conflicts can be detected. Also, the completeness of the list of update rules has to be proven.

A further issue would be to check, if the status values and update rules can be utilized for determining tasks that can be executed next (the part of the tool, that is assumed to be given at the moment). Perhaps this can be achieved through a checking like this: "If task $A$ is executed next, then the constraint $opt(B) \geq man(B)$ (for an arbitrary task $B$) is violated". Thus, $A$ cannot be suggested now (in the next step) for execution. Automata like in [4] would not be needed in that case.

In the context of log-based recommendations, it could also be interesting to include reviews into the decision support system. Users could rate their decisions at some time after their execution which is valuable information in future. To improve the performance, once calculated status values could be stored as tables together with the respective history in a hash-based repository.

## ACKNOWLEDGEMENT

The work of Michael Heinrich Baumann is supported by a scholarship of "Hanns-Seidel-Stiftung (HSS)" which is funded by "Bundesministerium für Bildung und Forschung (BMBF)". The authors want to thank Lars Ackermann and Stefan Schönig, both with University of Bayreuth.

## REFERENCES

[1] M. Scaife and Y. Rogers, "External cognition: how do graphical representations work?" *International Journal of Human-Computer Studies*, vol. 45, no. 2, pp. 185–213, 1996.

[2] S. Zugal, J. Pinggera, and B. Weber, "Creating declarative process models using test driven modeling suite," in *IS Olympics: Information Systems in a Diverse World*, ser. LNBIP, S. Nurcan, Ed. Springer Berlin Heidelberg, 2012, vol. 107, pp. 16–32.

[3] W. M. van der Aalst, M. Weske, and D. Grnbauer, "Case handling: a new paradigm for business process support," *Data & Knowledge Engineering*, vol. 53, no. 2, pp. 129 – 162, 2005.

[4] M. Pesic, "Contraint-based workflow management systems: Shifting control to users," Ph.D. dissertation, Technische Universiteit Eindhoven, 2008.

[5] W. van der Aalst, M. Pesic, and H. Schonenberg, "Declarative workflows: Balancing between flexibility and support," *Computer Science - Research and Development*, vol. 23, no. 2, pp. 99–113, 2009.

[6] M. Pesic, H. Schonenberg, and W. van der Aalst, "DECLARE: Full support for loosely-structured processes," in *Enterprise Distributed Object Computing Conference, 2007. EDOC 2007. 11th IEEE International*, 2007, pp. 287–287.

[7] M. Pesic, M. Schonenberg, N. Sidorova, and W. van der Aalst, "Constraint-based workflow models: Change made easy," in *On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS*, ser. LNCS, R. Meersman and Z. Tari, Eds. Springer Berlin Heidelberg, 2007, vol. 4803, pp. 77–94.

[8] M. Baumann, M. H. Baumann, and S. Jablonski, "On behavioral process model similarity matching: A centroid-based approach," 2015, preprint. [Online]. Available: https://epub.uni-bayreuth.de/id/eprint/2051 [accessed 2015-07-18]

 ISBN: 978-1-61208-432-9 