The Process Checklist Generator: Establishing Paper-based Process Support

Marcel Bankau, Michaela Baumann, Michael Heinrich Baumann*, Stefan Schönig, and Stefan Jablonski

University of Bayreuth,
Universitätsstraße 30, 95447 Bayreuth, Germany
{firstname.surname}@uni-bayreuth.de

Abstract. When enterprises are determined to introduce process management, they usually aim at IT system supported execution of processes. In constrast to this common tendency of process technology, we provide a straightforward, quickly viable alternative to IT-based process support at a reasonable effort: the Process Checklist. The paper-based scheme we introduce follows classical checklist concepts and builds upon the checklist idea in order to reach the same objectives as IT systems: task coordination, execution guidance, traceability. Therefore, the Process Checklist Generator (PCG) presented in this demo allows users to quickly transform process models given in the standard BPMN notation into Process Checklists. With this tool, we show how meaningful process support can be established quickly.

Keywords: Process Modelling, Process Checklist, Paper-based Process Execution, Step-by-Step Guidance, Demo Track

1 Introduction

In nearly all industries, process models are a common tool to provide description, standardization, and execution support of complex applications in management, IT, production, etc. Besides classical workflow management systems (WfMSs) [1], a new and simple way of business process execution support for mainly human-driven, high-level processes, e.g., in hospitals or administration, within one company was established a few years ago: the *Process Checklist* [2], for which an example is given in Fig. 1. Common checklists are often used as reminders only [3] and are, due to their construction, not suitable for process guidance [4]. However, Process Checklists extend the concept of checklists in a way that paper-based step-by-step guidance through process models is achieved [2, 5]. Each Process Checklist grounds on an approved process model, but is independent of any IT-based WfMS. The checklist in Fig. 1 shows seven checklist points of two different kinds. Operating points (e.g. point 1) contain a list of

* The work of M. H. Baumann is supported by a scholarship of "Hanns-Seidel-Stiftung e. V. (HSS)," funded by "Bundesministerium für Bildung und Forschung (BMBF)."

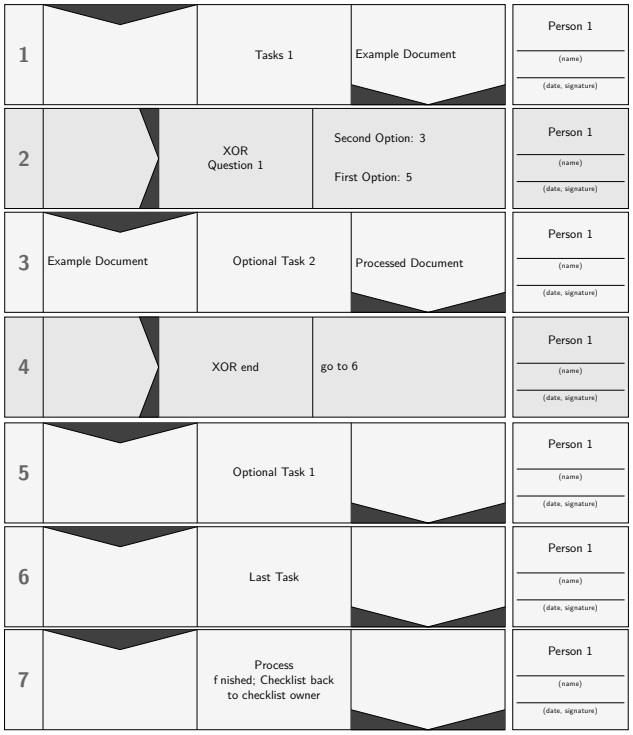

Fig. 1. The Process Checklist (without header) for the process model shown in Fig. 3.

incoming documents, an activity description, a list of outgoing documents and a field with responsible roles, where a specific executor has to sign. Control points (e.g. point 2) contain a list of needed documents, an instruction for a decision or parallel execution and pointers to subsequent subprocesses as well as a role field. By Baumann et al. [5], Process Checklists are motivated and defined, common checklists are discussed in detail, the transformation of Business Process Model and Notation (BPMN)[1] models into Process Checklists and their enactment are described, and their power of applicability is evaluated. The work at hand focuses on the automated creation of paper-based Process Checklists. For this, the *Process Checklist Generator (PCG)* is presented. The PCG uses BPMN models, like the example model of Fig. 3, in its XML representation to set up a LaTeX file from which a PDF is created.

2 Overview and Demo Guidelines

The PCG is a simple tool that transforms a BPMN model to a sequential list of tasks. A procedure of the performed steps is given in Fig. 2 taking the left path. In

[1] Business Process Model and Notation 2.0, `http://www.bpmn.org`, access: 2017-06-05

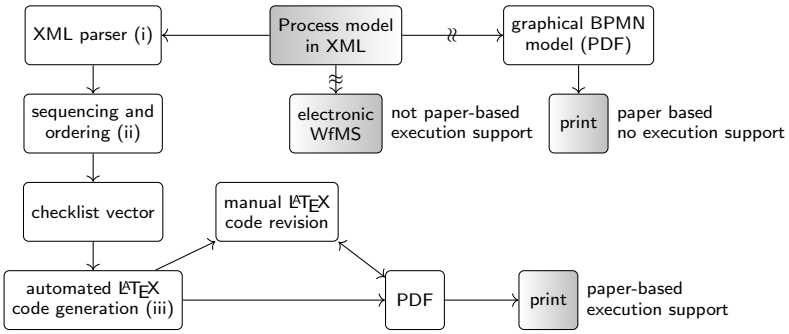

Fig. 2. From an XML BPMN model to execution.

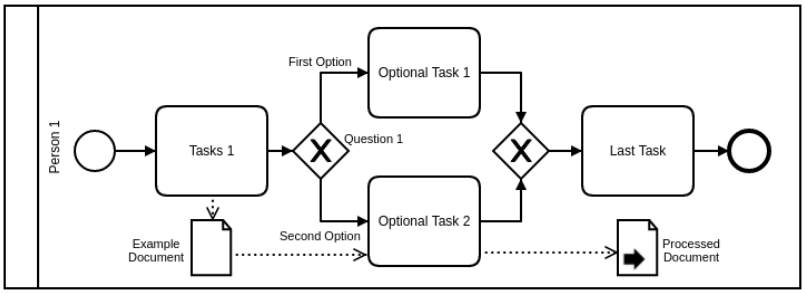

Fig. 3. Example BPMN model

the following, we provide an overview of the way of working and the results of the PCG. The exemplary process shown in Fig. 3 is the basis for the demonstration. Even though the process model is very small, the generated Process Checklist will provide the reader a general understanding of its functionality. The demo UI needs the path to the BPMN file as input from the user. The user also specifies the name of the output PDF file and the name of the Process Checklist. The Process Checklist itself is structured as proposed by Baumann et al. [2, 5]. The Process Checklist generated from the model in Fig. 3 is shown in Fig. 1. The program is written in Python and the PDF is generated with LaTeX. The procedure of the Process Checklist generation can be divided into three major parts:

(i) the parsing of the BPMN file
(ii) the sorting of the order of the Process Checklist items
(iii) the (partially) automated generation of the LaTeX file

When parsing the BPMN file, all information from the file is stored in memory. Amongst others, this includes the names of the participating agents, all tasks and gateways, their edges, and specific events. Even though the BPMN model language has a specific syntax, the process models used for this demo

were exclusively built using the Camunda Modeler[2]. Since the parsing is heavily dependent on the XML tags, it is questionable whether models generated by other tools might parse correctly.

During the second part of the generation program, the order in which the tasks should be performed is determined. The sorting algorithm starts to queue the sequential tasks beginning from the START event until a split gate is encountered. In the example process model from Fig. 3 this would correspond to the node after the task "Tasks 1". Once such a node with multiple outgoing edges is reached, the algorithm determines the corresponding join gate, which has multiple incoming edges. The Process Checklist item for a split gate contains information for the user where to jump next. This is depicted in Fig. 1, Point 2. The order in which the tasks inside this gate pair are queued is ordered path by path. This means the algorithm (arbitrarily) chooses the outgoing edges from the gate one after another and queues the tasks on the respective paths. This process is iterative, which allows us to generate Process Checklists for nested gates. Besides the order of the tasks explicitly mapped by the process model, we insert additional steps into the Process Checklist. These steps help the user to navigate through the Process Checklist. For example, we insert END points after finished paths between gate pairs (Fig. 1, Point 4). The point contains a GOTO instruction where the user has to continue with the Process Checklist. Further, an END point is added to the Process Checklist to signal the user that the Process Checklist is completed. If the process model has a case where the next designated task was already performed, the Process Checklist gets another additional node where the user is required to restart the Process Checklist beginning with the original next task. For example such a case could be encountered if a proposal needs external approval and has to be revised and resubmitted if not approved (cf. [5], Sec. 4.5.1). In the third and final step, all tasks are written to a LaTeX file using a LaTeX library for Python[3]. Each different type of node (i.e., tasks, events, gates, ...) have their own LaTeX template. These templates dynamically compose the boxes (as shown in Fig. 1) using the stored information about the nodes. For example a split XOR gate has, besides the description, all different path choices listed (Fig. 1, Point 2). Clearly, each choice is accompanied by the number where the user has to continue the Process Checklist.

3 Conclusion, Maturity, and Future Work

Most process models are stored digitally and their execution is possible only via electronic and cost-intensive workflow management systems. Printed versions are graphical representations only and therefore not suitable for process execution, e.g., they do not provide any step-by-step guidance. The presented PCG is a simple way to transform digital (resp. graphical) information about a process model into an easy-to-use paper-based Process Checklist. Even though the tool works quite well, some improvements could be done: Since process models can

[2] Camunda Modeler, `https://camunda.org/bpmn/tool`, access: 2017-06-05
[3] PyLaTeX, `https://github.com/JelteF/PyLaTeX`, access: 2017-06-05

contain quite large procedures, it is useful for the user to be able to split large process models into multiple shorter Process Checklists. This feature could be added to the implementation by providing the user with choices where to split the Process Checklist after the parsing of the BPMN file. Another improvement, which concerns the AND gates, would be to implement the dynamic sequential transformation proposed in [2, 5] to determine an execution order of parallel paths at runtime. A second area of improvement is the extension of the set of supported events, since now only the most important events can be mapped. However, not all kinds of events make sense in the context of Process Checklists, so more theoretical work and interviews with practitioners have to be made. At the moment, the PCG supports input files from the Camunda Modeler only. This is caused (even though the BPMN language is standardized) by slightly varying syntaxes used by different modeling software. It would be desirable to improve the robustness of the parser by including a wider range of XML tags. Also, experiences on the use of the tool in real life settings need to be acquired to learn other potential improvements.The fundamental functions of the generation algorithm are sound. However, the graphical presentation of the Process Checklist is not yet optimized, e.g., the number of displayed options in control points is limited. In extreme cases, boxes may be overfilled. Currently, we add redundant checklist items to the model. For instance, the last END node for a XOR gate is not needed, since the user can simply continue with the next task (see Fig. 1, Points 5 and 6).

The usage of the PCG is very easy and straight forward. It can be used by either a GUI or the command line. This offers an easy-to-use method for normal use-cases, but also a way to integrate the PCG into automated workflows. In general, the PCG is operated by one user. The PCG is available as an executable file as well as a python package. More information and install instructions as well as a screencast demonstrating the usage of the PCG is available at `http://checklists.kppq.de`.

References

1. van der Aalst, Wil, and van Hee, Kees Max: *Workflow management: models, methods, and systems.* MIT press. (2004)
2. Baumann, Michaela, Baumann, Michael H., Schönig, Stefan, and Jablonski, Stefan: Enhancing Feasibility of Human-Driven Processes by Transforming Process Models to Process Checklists. In: Enterprise, Business-Process and Information Systems Modeling, 124-138. Springer Berlin Heidelberg. (2014)
3. Wolff, Alan M., Taylor, Sally A., and McCabe, Janette F.: Using checklists and reminders in clinical pathways to improve hospital inpatient care. In: Medical Journal of Australia 181, 428-431. (2004)
4. Reijers, Hajo A., Henrik, Leopold, and Recker, Jan: Towards a Science of Checklists. In: Proceedings of the 50th Hawaii International Conference on System Sciences. (2017)
5. Baumann, Michaela, Baumann, Michael H., Schönig, Stefan, and Jablonski, Stefan: The Process Checklist. In: Enterprise Modelling and Information Systems Architectures, 12, 1-1 (2017)