The Process Checklist

Paper-based Enactment of Human-driven Processes

Michaela Baumann[*,a], Michael Heinrich Baumann[b], Stefan Schönig[a], Stefan Jablonski[a]

[a] Institute for Computer Science, University of Bayreuth, Germany
[b] Institute for Mathematics, University of Bayreuth, Germany

Abstract. *When enterprises are determined to introduce process management, they usually aim at IT system supported execution of processes by Workflow Management Systems (WfMSs) or Process-aware Information Systems. In constrast to this common tendency of process technology, we introduce a paper-based scheme to enact and execute human-driven processes in the work at hand. Our approach is motivated by insights into problems of firms that tried to establish process technology and failed with conventional methods. One of the design objectives for our scheme was to provide a straightforward, quickly viable alternative to WfMS-based process execution at a reasonable effort. The paper-based scheme we introduce follows classical checklist concepts and builds upon the checklist idea in order to reach the same objectives as WfMSs: task coordination, execution guidance, traceability. In this article, we describe how to transform Business Process Model and Notation (BPMN) process models into Process Checklists. We also present extensive evaluations of this approach both in the academic and in the business domain.*

Keywords. Process Modelling • Process Checklists • Paper-based Process Execution

Communicated by S. Strecker. Received 2015-09-03. Accepted after 3 revisions on 2017-02-07.

1 Introduction

For approximately 20 years, process management has been regarded as an important operational task responsible both for the description of complex applications and for supporting their execution (Jablonski 2010). In traditional approaches, business processes are executed by Workflow-Management Systems (WfMSs) (Zairi 1997), also called Process-aware Information Systems or Process Engines. The key benefits of the application of WfMSs are task coordination, step-by-step guidance through process execution and traceability supporting compliance issues (Reichert and Weber 2012). Nevertheless, it also has to be taken into account that the introduction of a WfMS is a time consuming and cost intensive task requiring profound technical skills. In particular, this research is motivated by practical problems that arose from a project with several small and medium-sized enterprises (SMEs) and the local Chamber of Industry and Commerce, *IHK Oberfranken*. The project C^2P^2 revealed that despite the wish to introduce business process management (BPM) technology, the fear of not being able to cope with the new IT system technology inhibits their introduction (C2P2 2015). We identified several

* Corresponding author.
E-mail. michaela.baumann@uni-bayreuth.de
The authors gratefully acknowledge the superb collaboration with *Sparkasse Bamberg*. Especially, the authors would like to thank Stephan Kirchner, board member and CEO of *Sparkasse Bamberg*. In addition, the authors thank all reviewers and editors, especially Stefan Strecker, for their valuable comments and Jonathan Owens and Lars Ackermann for their support. The work of Michael Heinrich Baumann is supported by a scholarship of *Hanns-Seidel-Stiftung e.V. (HSS)*, funded by *Bundesministerium für Bildung und Forschung (BMBF)*.

reasons why organisations refrain from introducing WfMSs. A first one is that they consider the introduction of a WfMS as a non-affordable knowledge- and cost-intensive IT project (Melenovsky 2005). This is especially applicable for SMEs. For instance, Chong (2014) mentions lack of financial resources and lack of time as the two most inhibiting factors for IT-driven BPM technology in SMEs. Also the GPM Netzwerk, an association that promotes the introduction of business process technology in Germany, identified cost and lack of time but also lack of expertise as factors that prevent the introduction of IT-driven BPM technology in SMEs (GPM Netzwerk 2015). These findings conform to our experiences within the C^2P^2-project.

Thus, we raise the question: Is there an alternative way to leverage on process management technology that avoids the previously named efforts but still maintains the key benefits of WfMSs? Of course, this alternative might yield the need for a different implementation of these benefits.

In addition to the constraint to certain organisations introducing WfMSs noted above, we also came across other reasons not to rely on IT-based BPM technology. One was the missing flexibility of such systems (Montali 2009; Zeising et al. 2014). For business processes that frequently depend on dynamic human decisions, conventional WfMSs turned out to be too restrictive (Swenson 2010; van der Aalst et al. 2005). In particular, these systems do not appropriately deal with exceptions, which regularly occur in human-driven workflows. Often, the only way to deal with an exception is to bypass the system. This observation yields the general discussion about 'the computer won't let them', i.e. that a computer does not allow doing things users like to do (Condon 1993).

Besides, we can follow Luff's argument that if (original) paper documents are needed for process executing, in many cases a paper-based execution tool is also preferred (Luff et al. 1992). However, there already exist decent approaches that cope with such situations. Nevertheless and all in all, despite the pervasiveness of enterprises by IT technologies, especially in the BPM area, the additional introduction of 'another IT-system' has to be regarded as critical.

As an alternative way of supporting the execution of processes independent of IT-based process management systems, we propose a paper-based scheme. This new and unconventional way of process execution support is especially suitable for human-driven processes, i.e. for such processes that cover, according to Harrison-Broninski (2010), rather high-level, especially management work, knowledge work with a certain degree of autonomy of the involved agents or critical human actions such as the health care sector. Further, the paper-based scheme is particularly appropriate for well-structured processes with a relatively small degree of creativity but with a high degree of flexibility in the sense of dynamic human decisions as well as fast and uncomplicated reacting to external circumstances as, for example, is usually required in clinical processes. Thus, for those shunning the introduction of WfMSs or for those in need of a certain type of flexibility within their rigid processes, we introduce the so-called 'Process Checklist'. By this approach, we want to reduce the necessity of IT systems for BPM to a certain extent. The starting point of the approach is the modelling of a business process model, obtained, for example, through the use of an IT-based process modelling system. This is justified since a process modelling system is—from a technical view—much simpler to run than a process execution system. In addition, often a process modelling system can be obtained free of charge (see, for example, AXONiVY 2016). In contrast, a process execution system often requires a complex installation and typically is not free.

Based on this business process model, we describe a transformation algorithm to obtain the Process Checklist for which we define the general structure and the enactment, i.e. how to execute a process with the help of the Process Checklist. This means, we replace an IT-based process execution system by a paper-based step-by-step instruction manual. The Process Checklist is physically handed over from process participant to process participant during process execution.

Thus, the implementation of such a process execution system is virtually nonexistent; IT skills and knowledge are not needed.

Despite being entirely paper-based, a Process Checklist still supports key benefits of IT-based WfMSs. The checklist is handed over to responsible agents (task coordination), process tasks are serialised and marked by a unique identifier (step-by-step guidance) and the checklist itself as well as the corresponding signatures ensure traceable process execution that can be archived at the end without the need for a special support.

The work at hand introduces the general structure of Process Checklists as well as an elaborate transformation algorithm of basic Business Process Model and Notation (BPMN) diagrams Object Management Group Inc. 2011 to Process Checklists by giving transformation instructions for the particular process model elements. The results were basically achieved through a design-oriented research see Hevner and Chatterjee 2010, characterised through seven guidelines for design science like problem relevance and design as a search process, as we first faced the problem of finding a fast and easy process execution support tool meeting all the requirements of a proper process model. After investigating existing methods, we developed the Process Checklist on the basis of the available checklist designs and supplemented them with the necessary process model elements. After that, we validated the prototype checklist with requirement comparisons and use cases. The design-oriented research approach suited our goal as we started form the practical problem that we wanted to solve. Note that this work is based on Baumann et al. (2014). In addition to various improvements and further concepts, the work at hand extends by a description of how it is actual enacted (Sect. 6) as well as by a detailed evaluation and case studies (Sect. 7).

2 Structure of the Process Checklist

According to Jablonski and Bussler (1996), each process model should cover at least five perspectives. Accordingly a Process Checklist should therefore support the following to serve as a sophisticated support tool for process execution: There are

- the functional perspective (i.e. the task description),
- the organisational perspective (i.e. the assignment of agents to tasks),
- the data perspective (i.e. the description of data flow, including data generation and consumption),
- the operational perspective (i.e. the assignment of systems and services to tasks) and
- the control flow perspective (i.e. the order of tasks to follow in an execution).

The Process Checklist presented in this work is designed in compliance with these perspectives. All of them are present in the Process Checklist in Fig. 1.

In principle, we distinguish two kinds of checklist steps, also termed checklist points. There are operating points, which describe the activities of a process, and control points, which decide about the execution order of the activities. The different points are indicated through different colours in the example of Fig. 1. Operating points are light coloured and control points are dark coloured (in Fig. 1, Steps No. 2 and 8). The two different kinds of checklist points are explained in the following.

A schematical representation of an operating point is given in Fig. 2. An operating point has five fields that possibly need to be filled in with information about a certain activity. Field i on the left-hand side assigns a unique number, i.e. unique for one single checklist, to the point that is needed to address this point from within other checklist points. Field AC_i in the middle contains the description of the activity (AC) that has to be fulfilled, possibly also containing information about the system or service that has to be used when fulfilling the task. Field ID_i on the left of the activity description contains a list of incoming data (ID), i.e. consumed data and documents that are needed to perform the activity, and field OD_i on the right of the activity description contains a

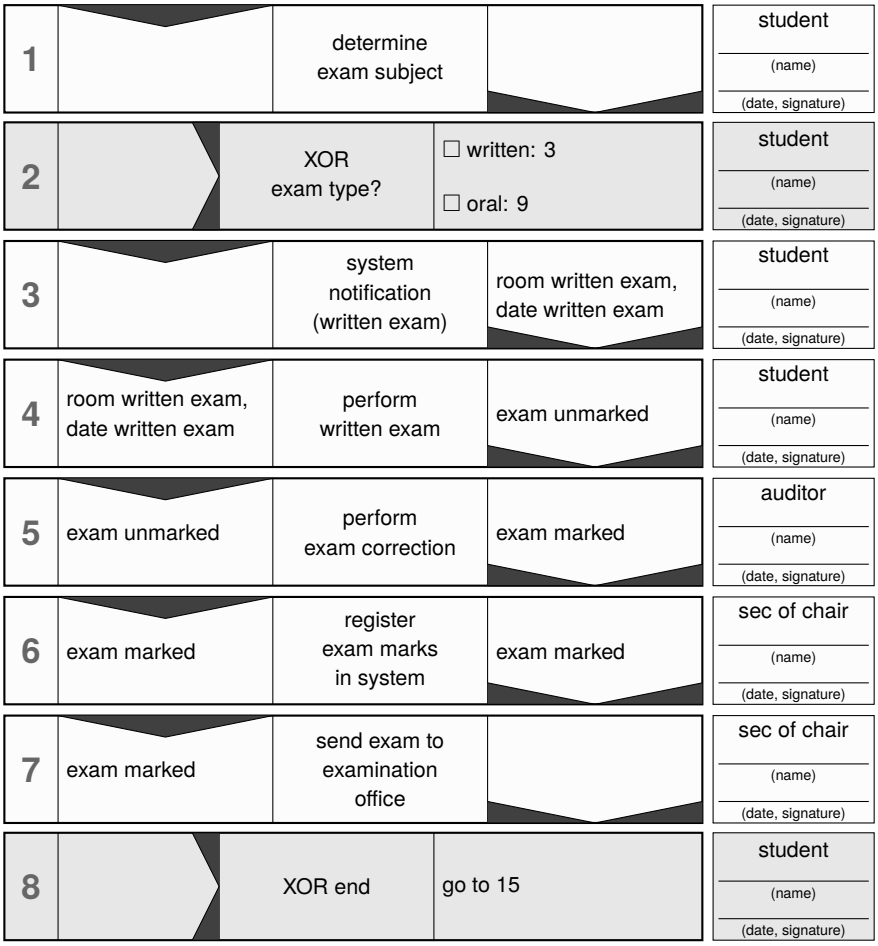

Figure 1: First part of the Process Checklist for the process 'Administering an exam'.

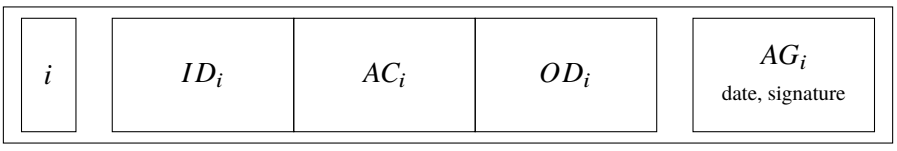

Figure 2: Schematical representation of an operating point.

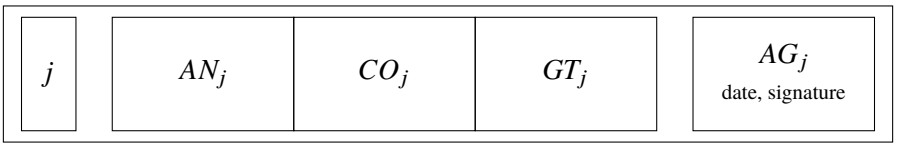

Figure 3: Schematical representation of a control point.

list of outgoing data (OD), i.e. data and documents that are produced during the activity. It is also allowed to list documents in ID_i and OD_i that are only passed through and not needed for fulfilling task AC_i. Field AG_i assigns a human agent (AG), or in general a role, to the activity. Altogether, an operating point contains information about the functional (field AC_i), organisational (field AG_i), operational (potentially mentioned in field AC_i) and data-flow perspective (fields ID_i and OD_i). The numbers on the left of each point serve as identifiers but do not basically impose an execution order of the points. This is mainly done by the second kind of checklist points, the control points. A schematical representation of a control point is given in Fig. 3.

A control point consists of five fields, too. Again, on the left-hand side, there is a unique number j assigned to the point. The uniqueness of the point numbers applies jointly for operating and control points, so if there is an operating point with number k, there is no control point with number k and vice versa. Field CO_j in the middle contains advices for splitting and joining the control flow and, when splitting with respect to several alternatives, also contains the condition (CO) or question under which one or more alternatives has or have to be chosen. Field AN_j is reserved for annotations (AN) that can contain instructions for documents when needed to check the condition in CO_j or a recommended execution order of independent alternatives.

In field GT_j, the go-to instructions for the different alternatives are listed. A refinement of field GT_j is shown in Fig. 4. Attributes $a_{j,\cdot}$ represent the answers (a) to the question in field CO_j or concrete forms of the condition that has to be checked. If field CO_j indicates subprocesses that may be executed independently, then attributes $a_{j,\cdot}$ are simply replaced by the phrase "go to". Also, the names of subchecklists, which will be explained later, may be listed there. Attributes $g_{j,\cdot}$ refer to the numbers of other checklist points, where the checklist has to be passed on in the corresponding cases. These points can either be operating or control points. In the sample checklist in Fig. 1, an exam type has to be chosen exclusively in control point No. 2. In control point No. 8, one subprocess of an exclusive split is finished. The variable assignment in control point No. 2 for field GT_2 is as follows: $a_{2,1}$ = written, $g_{2,1} = 3$, $a_{2,2}$ = oral, $g_{2,2} = 9$. The confirmation lines (date, sign.) behind every (a, g)-pair as shown in Fig. 4 are not visible here because the exclusive control point is performed only once. Whether the confirmation line is shown or not is apparent through a Boolean attribute c, which is $c_2 = 0$ in the case of Fig. 1. Why this confirmation line, the last (a, g)-pair $a_{j,k}, g_{j,k}$ without box $\square$, that may also be not visible if $a_{j,k}$ = "" and $g_{j,k}$ = "", and all other components are needed, will be further explained in Sect. 4. In short, field GT contains information about how to navigate through the checklist when processing exclusive or inclusive choices or parallelism. Parallelism means that certain independent subprocesses not influencing each other may be executed at the same time. If this is not possible, i.e. if the subprocesses have to be executed by the same agent where concurrency is a physical problem, their order of execution is not prescribed.

To conclude this section, a fully formalised representation of the Process Checklist, making use of the field names introduced above, is available in Definition 1. Every checklist is depicted by a vector of operating points p^o and control points p^c.

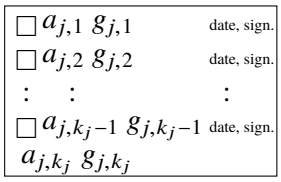

Figure 4: Schematical representation of field GT_j of control point No. j.

Definition 1 (Checklist Vector) A checklist is a vector $C = (p_1^t, p_2^t, \ldots, p_n^t), n \in \mathbb{N}, t \in \{o, c\}$ with two different kinds of components:

$$p_i^o = (ID_i, AC_i, OD_i, AG_i) \quad \text{(operating point)}$$

with ID_i, AC_i, OD_i, AG_i being strings and

$$p_j^c = (AN_j, CO_j, GT_j, AG_j) \quad \text{(control point)}$$

with AN_j, CO_j, AG_j being strings and GT_j being a vector of the form

$$GT_j = (c_j, a_{j,1}, g_{j,1}, a_{j,2}, g_{j,2}, \ldots, a_{j,k_j}, g_{j,k_j})$$

with $k_j \in \mathbb{N}$, strings $a_{j,l}$, integers $g_{j,l} \in \{1, \ldots, n\} \cup \{""\}$, $l = 1, \ldots, k_j$ and $c_j \in \{0, 1\}$.

The checklist vector representation of the sample process 'Administering an exam' of Fig. 1 is given in Fig. 5. The first eight rows (No. 1 to No. 8) correspond to the graphical checklist of Fig. 1. In the following section, the Process Checklist design and structure shall be compared to already established checklist types.

3 Background and Related Work

Despite an extensive literature search, we were not able to find an accurate and universal definition of checklists. However, in a common understanding, a checklist is a list of required items, things to be done or points to be considered, usually used as a reminder (Wolff et al. 2004). Checklists are generally seen both as a helpful tool in daily life, e.g. when talking about shopping lists or packing slips, and as a suitable means for error management and performance improvement in highly complex scenarios like clinical workflows (Hales and Pronovost 2006), aircraft preparation (Degani and Wiener 1991; Hartel and Chou 1995) or project management (Boehm 1991), to mention only a few examples. Usually, in all of these fields checklists are rather an unsorted list of application specific items that have to be checked for validity. Routing slips or dockets are an exception in this context and are addressed, too. In the following a short introduction to the use of checklists in the above listed application fields shall be given to get a better idea of this topic.

3.1 Clinical (Symptom) Checklists

In the field of clinical healthcare, checklists, mostly paper checklists fixed on clipboards, are used for detecting or determining diseases (e.g. Briere et al. 2001; Derogatis et al. 1974). Roughly summarised, symptoms are manually recorded according to a list of possible symptoms and then, analysing the checkmarks, a possible illness is ascertained. In this case, the checklist is a means that helps physicians to detect a patient's disease but it does not provide information about the process of analysing the patient's health condition itself. Faerber et al. (2007) propose the use of checklists to compactly map small process steps that can be executed in arbitrary order within a process model. This implies that this is a very simplified application of our approach as there is no ordering given for the process steps, no support of data flow and of organisational issues.

The idea of clinical checklists is further developed by the WHO (2009). There, a checklist is suggested to improve save surgery, i.e. to guide and monitor a whole operation team through a surgery process. This kind of checklist application is close to a Process Checklist we aim at, but there are two main points that make it unsuitable for our purpose. The first one, also mentioned by the WHO (2009), is that only one person is responsible for the checklist, i.e. the checklist can only be gone through and filled in by a single person. Furthermore, control flow including exclusive, inclusive or parallel execution orders, which are essential for business applications, are not available.

No.	type	*ID/AN*	*AC/CO*	*OD/GT*	*AG*
1	*o*		determine exam subject		student
2	*c*		XOR exam type?	$c_2 = 0$ $a_{2,1}$ = written $g_{2,1} = 3$ $a_{2,2}$ = oral $g_{2,2} = 9$ $a_{2,3}$ = "" $g_{2,3}$ = ""	student
3	*o*		system notification (written exam)	room written exam, date written exam	student
4	*o*	room written exam, date written exam	perform written exam	exam unmarked	student
5	*o*	exam unmarked	perform exam correction	exam marked	auditor
6	*o*	exam marked	register exam marks in system	exam marked	secretariat of chair
7	*o*	exam marked	send exam to examination office		secretariat of chair
8	*c*		XOR end	$c_8 = 0$ $a_{8,1}$ = go to $g_{8,1} = 15$	student
9	*o*		system notification (oral exam)		student
10	*o*		determine and assign examination date	examination date	secretariat of chair
11	*o*	examination date	perform oral exam	minutes of examination (unsigned)	auditor
12	*o*	minutes of examination (unsigned)	sign minutes of examination	minutes of examination (signed)	assessor
13	*o*	minutes of examination (unsigned)	sign minutes of examination	minutes of examination (signed)	auditor
14	*o*	minutes of examination (signed)	send exam mark and protocol to examination office		secretariat of chair
15	*o*		exam notification and performance finished		secretariat of chair

Figure 5: Checklist vector for the process 'Administering an exam' containing all elements according to Definition 1 needed for the graphical checklist.

3.2 Aircraft Crew Checklists

Checklists in the field of air traffic mainly serve as a reminder to obtain a proper configuration of the plane and full quality and security in every flight (Degani and Wiener 1991; Hartel and Chou 1995). They are important especially for enhancing the coordination during high workload and stressful conditions, but also to reduce variability between pilots. Throughout the years, they have transformed from a simple memory-aid to a task by themselves. The most common type of checklist is a paper checklist as it is a very simple device that may be held by the pilot, clipped to the yoke or glued to instruments. As Degani and Wiener (1991) mention, there are also several disadvantages to paper checklists. The main one is the lack of a pointer to distinguish between accomplished and non-accomplished items, but also the lack of a distinction between unaccomplished items that are not yet done and that will not be done, the need to hold checklists in one hand, meaning one hand is occupied by the checklist, or the difficulty to read them at night. This is why several other types of checklists are common in the field of air traffic. However, all types of checklists that are mentioned in this context lack some of the process perspectives listed at the beginning of Sect. 2. Apart from the functional perspective, no other perspectives are fully visible on aircraft paper checklists. By Degani and Wiener (1991) also some guidelines for designing and using flight-deck checklists are listed, like the consideration of the workload of human agents when assigning the different tasks and the ordering of checklist items either according to certain dependencies or arbitrarily. As far as possible, we set up our checklist approach with a view to these guidelines. However, the psychological aspect and mental factors that are mostly related to the strong responsibility a pilot has, were not of great importance to us as to Degani and Wiener (1991).

3.3 Project Management Checklists

Checklists in the field of project management can either help organising the project team members or serve as identifiers, e.g. risk identifiers (Boehm 1991). These identifier checklists are similar to medical checklists, which means that they do not provide information about a procedure itself but operate more like a decision support (Kerzner 2013). Checklists helping organising the team are task lists that allow assigning team members to the task and distinguishing between mandatory and optional items (Kerzner 2013). These task lists are however customised to each project, rarely cover whole processes and do not follow any standardisation. They also do not contain information about the five process perspectives except for the functional one. But they serve as a good starting point, together with the other presented types of checklists, for the Process Checklist representation and its features formulated in the work at hand.

3.4 Routing Slips/Dockets

Routing slips are already a more sophisticated type of checklist used in the context of BPM and process execution. They provide information not only about the task itself but also about participating agents, interfaces, process branching as well as processing, transport, layover and throughput time (Organisationshandbuch 2015). The Organisationshandbuch (2015) describes the way of how to use a paper-based routing slip whereas Kumar and Zhao (2002) explain an electronic one. Basically, a routing slip records the several steps within a process with start and end time and signatures of the executing agents. One great disadvantage of routing slips is that they are always attached to a physical or electronic document. Therefore, they only monitor the course of a specific document and cannot be used across documents. Our checklist approach extends the concept of routing slips not only in this document-bounded context but also allows for a number of elaborate process patterns not representable on common routing slips, like parallel subprocesses, iterations or role assignment. In the Organisationshandbuch (2015), routing slips are recommended as a means for identifying, not executing, processes and process variants during the modelling phase. One use case for analysing processes with routing slips in the healthcare sector is described by Kellerhoff (2012).

3.5 Contribution and Delimitation to Related Work

In this paper, we try to bring the widely used unstructured checklists, that usually do not provide any information about execution order and other execution modalities like parallelism or exclusiveness, as well as the concept of routing slips into a more structured and generally applicable form to receive a structured process execution tool, but also to maintain the uncomplicated usage of paper-based checklists. By Jablonski (2010) and Seitz et al. (2014), this kind of checklist is suggested but not elaborated on. The disadvantages listed in Sect. 3.2 are either not essential when it comes to Business Process Checklists (like one hand being busy) or are dissolved by special checklist features, e.g. the problem of a missing pointer is resolved by the signing field *AG*. The structure of a Process Checklist as proposed in the work at hand still reveals similarities to the traditional checklists briefly presented above, especially to routing slips, but also gives room to the process part of the desired process execution tool. The traditional task lists (e.g. the pilots' aircraft configuration checklists) are somehow represented by the operating points where their execution can be confirmed by signing them. Of course, every agent has its unique identifier to guarantee traceability of every process execution. Simple checkmarks are not enough to achieve this issue. In the following, the execution of a Process Checklist consisting of the elements presented in Sect. 2 is explained. Linked to the execution of a checklist is the generation of a checklist. Of course, a checklist has to be generated first before it can be executed, but generating a Process Checklist also requires knowledge about its execution. At some points, a certain execution method of the checklist requires a certain design of the checklist, which has to be considered at generation time.

Theoretic approaches of structuring workflows, excluding business process models that emphasise control-flow, can be found in the field of structured programming. There, structograms, like the Nassi-Shneiderman diagram (Nassi and Shneiderman 1973) or the Jackson structured programming (Jackson 1975), are used to graphically outline a programme's structure. However, they are more similar to graphical business process models but lack the possibility of expressing parallel execution without conditions and arbitrary backward jumps, as well as assigned agents and a separate data flow. Also, the presentation did not seem appropriate to us to serve as a support tool that is swift and easy to understand. This is why we did not examine structograms any further.

The main differences that distinguish the Process Checklist from the related approaches are the following:

- The basis for the Process Checklist is a commonly agreed process model.
- It is clearly defined which elements are needed to generate a checklist applicable as process execution support.
- The Process Checklist is able to fully reflect the original flow of the process, including exclusive, inclusive and parallel subprocesses and other relevant process patterns.
- The Process Checklist involves multiple users and coordinates their activities according to the underlying process model.

These differences constitute the novelty of the Process Checklist.

4 Enactment of the Graphical Checklist

The checklist method describes a valuable form of process usage and widens its spectrum towards non-computer based and flexible process execution where traceability is still maintained. However, before turning towards the execution of the single Process Checklist elements, an additional component besides the graphical checklist itself is required, a so-called 'cover sheet'. A cover sheet identifies a process instance. The cover sheet contains a timestamp, a text field for important information about the process, the name of the process owner, i.e. usually the person that started the process instance and that serves as contact person for this process instance, and a table showing

the current state of the process. Depending on the process, other information may be provided, too. The state indicator is just a list of already accomplished checklist points addressed by their numbers and one point that is being processed at the moment. When the process contains loops, the number of the current checklist is noted directly before the numbers of the checklist points. This list of points serving as state indicator induces a pointer, which is listed as one of the missing checklist elements by Degani and Wiener (1991). The cover sheet and the graphical checklist together form the basis of the Process Checklist bundle, which is, in this form, ready for execution.

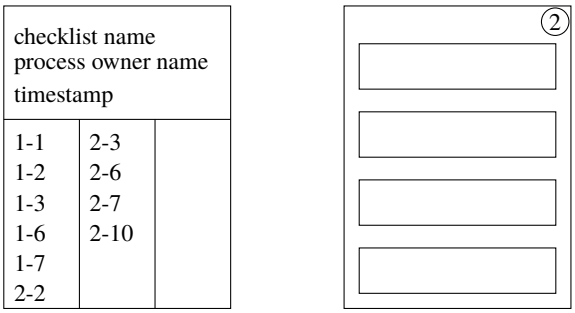

Figure 6: Cover sheet (left-hand side) with name of the checklist/process, name of process owner and a list of already executed steps and one point to be executed next; checklist (right-hand side) with current number in the upper right corner and several operating/control points. Note that in this fictional checklist with serial No. 1, a gateway caused a jump into the past (from point No. 7 to point No. 2), apparent on the cover sheet.

4.1 Traceability of Checklist Executions

Successfully accomplished tasks are recorded on the Process Checklist via signatures of corresponding human agents. One of the most important features is that at the end of the process all required signatures must be gathered on the checklist. If needed, certain parts of a checklist can even be deleted, changed or added during the execution by simply using a pen. To achieve traceability in the normal and in the modification case, not only the signatures but also another prerequisite is required: proper processing. Human agents have to be honest and conscientious to support this principle. That means, among other things, that each modification of a process template is signed by the corresponding agent. When modifying the Process Checklist, maintaining consistency becomes a critical duty. It can be addressed by incorporating milestones into the checklist where the process owner has the chance to review the Process Checklist. This solution is insufficient from a global IT point of view but for practical use it is enough. Additionally, reviewing by the process owner has deterrent effects since the executing agents consider deviations carefully. In our three sample scenarios presented in Sect. 7, it was no issue to enforce this regulation. The handling of exceptions is also briefly discussed in Sect. 4.3.

In some cases, there also may be more than one checklists attached to the checklist bundle as it is possible that some points need to be executed more than once. This is the case when iterations are necessary. Iterations are implemented by so-called backward jumps where the checklist is printed again and attached to the bundle. To distinguish between the several checklists, each of them gets a consecutive number starting with No. 1. This checklist number is not necessary if there is no backward jump possible during the process. An example for a checklist with backward jump is given in Fig. 6.

As one can see on the cover sheet on the left-hand side of Fig. 6, the enactment of the checklist started with point No. 1 (actually point No. 1 of checklist No. 1, indicated through 1–1), then 2, 3, 6 and 7 followed (1–2, 1–3, 1–6 and 1–7 as it is still the first checklist). After point No. 7, a backward jump was performed and the checklist number increased from 1 to 2. That means the checklist was printed a second time. Then, point No. 2 was executed a second time (2–2), as well as points 3, 6 and 7 (2–3, 2–6 and 2–7). The point to be performed next is 2–10, i.e. point No. 10 of the second checklist at the back of the Process Checklist bundle.

Actually, the cover sheet does not contain additional information except for the process owner, but it helps to quickly reproduce the process and

find the current point (the last one in the list of all points). What may be part of the checklist bundle, too, are the data objects handed over with the checklist. Also, the bundle can be extended by a list of data objects so that they can be checked for completeness and a receipt book so that the transmission of the checklist can be validated. The receipts filled out remain at the corresponding agents.

4.2 Execution Overview

When starting a process with checklists, the process owner, i.e. the person starting the execution of the process, has to print the checklist with cover sheet and data object list. Then he assigns the checklist its current No. 1. Input data, which means all input paper documents, have to be added and written in the respective list. On the cover sheet "1–1" is noted, meaning the current status of execution is 'checklist No. 1' and 'point No. 1'. In addition, he has to write his name on the cover sheet so that the checklist can be handed over to him after finishing the process. Additional information, like starting time of the instance, can be noted, too. The Process Checklist bundle has to be passed to the agent named in point No. 1, who has to check for completeness, most importantly to ensure that all listed documents are handed over, and quit the delivery. The process owner has to archive the signed receipt for later reconstruction if necessary. We assume here that the addresses of the participating agents are known or can be looked up in an address list. In SMEs this should, however, be no problem. The employees are mostly in the same building or the checklist can be sent via interoffice mail. If it is sent via mail, then the postman has to sign the receipt (if available).

When an agent gets the graphical checklist, he has to run through this acknowledgement process (check the documents for completeness, sign a receipt) and then check for the current point of the checklist on the cover sheet. When the last entry is 2–10 (as in Fig. 6) the operator has to look at point 10 of the current checklist, which has number 2, and execute this point if all necessary documents are available and possible conditions are fulfilled. Of course, the agent named in this point should be correct (otherwise the checklist has not been handed over properly). The corresponding agents can be found on the right-hand side of every point. A concrete name can be filled in by hand either by the executor of the last recent point, by the process owner at the beginning or, for example, by a department secretary that distributes incoming mail. After execution of the current point, the agent has to look which agent is next. If it is himself he executes the next point and writes it down on the cover sheet. Otherwise he updates the document list, writes the next point on the cover sheet, hands the checklist over and archives the received receipt. If one agent sends a document directly to another person, this document has to be deleted from the data object list and maybe listed again later on by the other agent.

4.3 Exceptions during Execution

Although this paper does not focus on an elaborated concept of exception handling for the Process Checklist, we will briefly discuss how this approach can well cope with exceptions during process execution. According to Reichert and Weber (2012), exceptions are traced back to sources like external events, activity failures, deadline expiration, resource unavailability and constraint violation. We are concentrating on unexpected exception events and assume that expected exception events are already covered by the process model.

The occurrence of an exception often results in one of the following situations, to name a few typical examples: a certain process step must be skipped, a new process step must be included or a process must be quit. All these scenarios have in common that the execution thread is interrupted and process execution continues in an unexpected way. We recommend the following exception handling policy for Process Checklist users. Firstly, the source of and the effect for an exception must be noted on the Process Checklist. Secondly, the current agent has to consult with the process owner about how to proceed in order to avoid arbitrariness. All decisions and

changes are recorded on the Process Checklist. In principle, the current agent and the process owner together can decide any kind of process continuation possible. In case the process owner becomes a bottleneck, since too many exceptions are occurring, another escalation strategy must be selected. Without detailing this discussion in this paper, we merely want to postulate that whatever strategy will be chosen, it must be guaranteed that exceptions cannot be taken haphazardly.

At first sight, the above described proceeding seems to be dissatisfactory and unstructured. However, it brings a couple of benefits. Firstly, there is always a way of continuation since the current agent and the process owner can define one. Secondly, the recorded information about the exception can be fed back to the process modeller to improve the current version of the process. In total, the Process Checklist is reasonably robust against exceptions whereby a comprehensive policy for exception handling still has to be elaborated.

4.4 Execution of Operating Points

Operating points are executed straightaway as described above, performing the task as given in AC. If documents are produced, they must correspond to the ones listed in the outgoing documents OD. After performing the task, the responsible agent signs the operating point to make clear he has finished this point. Operating points usually do not have go-to numbers, so the next point in the checklist is executed next.

4.5 Execution of Control Points

Control points can have different characteristics as they represent, for example, exclusive, inclusive or parallel splits. These three terms, which are presented in the following, describe the three ways that cover the common possible procedures appearing in business processes besides sequence flows: alternatives, loops and pure independent execution of several subprocesses. Apart from the control flow steering function, control points also serve as redirection advices that are necessary when certain points have to be skipped or for joining the distribution of multiple subchecklists.

4.5.1 Execution of Exclusive Splits

If a control point marked with the word XOR, which indicates a choice to be made where exactly one answer/condition is true, has to be processed, the agent has to check for the condition or question in field CO. He marks his answer in GT in the box $\square$ in front of the corresponding answer, e.g. the l-th answer $a_{\cdot,l}$. If there are any documents helping him to decide, they are listed in AN. After marking, he gets the number of the next point, $g_{\cdot,l}$. Two possible scenarios may occur. In the first, if $g_{\cdot,l}$ is greater than the current point number, then everything can go on as before, meaning $g_{\cdot,l}$ is the next point to be executed. In the second, if $g_{\cdot,l}$ is smaller than the current point number, then there is a problem, as that point with number $g_{\cdot,l}$ or other points may have been processed already in the past and therefore are signed already. If such a backward jump occurs, than the agent of the control point has to print a new checklist (just the checklist itself) and assign it the number $i + 1$ if the number of the current checklist was i. On the cover sheet, he writes for the next point to be executed $(i + 1)$–$(g_{\cdot,l})$. After doing this, he signs in field AG and passes the new checklist (together with the old one for reconstruction opportunity) to the agent of point $g_{\cdot,l}$. This agent has to recognise that the consecutive number of the checklist has changed, which is obvious on the cover sheet. Note, only with exclusive splits backward jumps are reasonable. Concerning other types of control points presented in the following, backward jumps would cause inconsistencies both in the model and the real world as, for example, a livelock would occur. When there are no backward jumps over the whole checklist, the consecutive checklist number is not needed.

4.5.2 Execution of Parallel Splits

If a control point marked with the word AND, which indicates a potentially parallel execution of at least two subprocesses, has to be executed, there are three possibilities to do so. (1) One method where only one checklist is needed is the dynamic sequential execution, which allows for

choosing a suitable order of the subprocesses. Parallelism, however, is dropped in this case. (2) The postbox method retains parallelism but relaxes the typical checklist properties. (3) Parallel execution also keeps parallelism but requires separate checklists for all subprocesses. The three methods are explained in more detail in the following.

Dynamic Sequential Execution

When coming to an AND control point that has listed the numbers of the starting points of several subprocesses in field *GT*, the agent of that point can decide about the execution order of the different branches during the processing of the checklist. He can take into account the current circumstances like availability of the agents in the different branches or anything else. An example for such a control point is shown in Fig. 7. Note that the last go-to number in field *GT* refers to the checklist point that has to be executed after all the subprocesses have been executed successfully.

When the agent chooses one branch, he marks his decision in the corresponding box □, notes it on the cover sheet and passes the graphical checklist over to the agent of the respective point. The branch is processed and at the end of this branch there is a control point that refers back to the control point where the decision for the branch was made. The order deciding agent therefore gets the checklist back (with checking for all documents and quitting again) and confirms the chosen and now successfully executed branch in *GT* (that one with the marked box that has not been confirmed yet). Then he chooses the next branch to be processed the same way as before. If all branches have been marked and confirmed in field *GT*, then he signs the whole control point in field *AG* and passes the checklist over to the agent of that point listed after "finally go to" in *GT*. The whole procedure can be reconstructed with the notes on the cover sheet.

Regarding the control point in Fig. 7, the following statements about the subbranches can be made: The first subprocess starts with point No. 10 and ends with a control point (unknown number; probably it is No. 11) referring back to the order decision point No. 9. The second subprocess starts at point No. 12 and again ends with a control point (unknown number; probably it is No. 17) referring back to point No. 9. After both subprocesses have been successfully executed, the branches are joined in point No. 18, which has to be executed no matter which order was chosen in point No. 9. Note that the control points referring back to the order decision control point do not induce backward jumps like such ones mentioned in the previous section about exclusive splits (Sect. 4.5.1). No operating point has to be executed more than once and the order decision control point is designed for being able to be executed more than once through the additional confirmation lines. Here, attribute *c* of the checklist vector definition is set to 1, i.e. the additional confirmation lines are visible.

Postbox Method Execution

The postbox method execution requires parallel splits designed in the same way as for the dynamic sequential execution. The difference is in the processing of the checklist, as the postbox method allows for parallel execution of the different branches, and the task assignment regime. When the processing of a checklist reaches a parallel split control node, the checklist is posted like an announcement in one place together with all documents (that can be stored in a postbox) and all agents can look for the next points that have to be executed on the cover sheet, where all first points of the different branches have to be noted in a parallel way, which could lead to a confusing cover sheet. Thus, the postbox method's control is pull-based by the operators of each subprocess, whereas the dynamic sequential execution is a push-based assignment by the operator of the control point. With the postbox method, the documents do not have to be handed over from one point to another but communication between the different agents, particularly of that ones involved in the same subprocess, is necessary to not waste time if two consecutive tasks have to be performed by different agents. After finishing all branches, the agent of the control node that started the postbox method collects the checklist and all

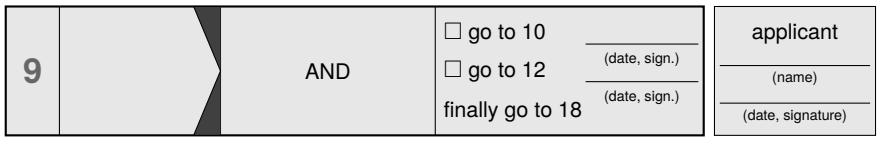

Figure 7: A control point suitable for dynamic sequential execution

documents now being in the postbox, checks for completeness, signs in *AG* if everything is okay and continues as before. This method may become confusing if too many agents are involved in the subprocesses and it needs coordination, initiative and especially individual responsibility of all agents. Although it provides a possibility of considering the parallelism aspect of the several subbranches, we generally do not recommend it as it is very difficult to realise. Responsibilities are not clearly assigned and consistency of the documents is even more difficult to maintain. The postbox method might be applied in a process with two participants where the checklist alternates between the two agents, e.g. during application procedures where the checklist is passed between the applicant and his superior (both located in the same building).

Parallel Execution

The parallel execution method enables the simultaneous execution of parallel branches. It requires a control node similar to that one of the dynamic sequential execution. But instead of referring to the initial points of the subprocesses in field *GT* in the same checklist, it is referred to the first points of separate subchecklists, one for every subprocess. The agent of the control point prints all required subchecklists, marks the boxes ☐ in *GT* if handed over together with needed documents to the respective agents of the first points in the subchecklists and confirms every returning subchecklist in *GT*. So again, variable $c = 1$. If all subchecklists have returned, he signs in *AG* and the execution of the control node is finished. The subchecklists are distributed in the same way as the main checklist: on foot or by (interoffice) mail. A finally-go-to number passes on to the next point of the main checklist. (This instruction is not necessarily needed, as the next point in the main checklist is executed either way if no separate jump instruction was given.) For this execution method of parallel splits, only one control point is needed in the main checklist, so the main checklist is probably less confusing than for the dynamic sequential execution. But as one can imagine, this method is more expensive as multiple checklists have to be generated. Nevertheless, it provides parallelism, i.e. a potentially faster execution of the checklist, and a good overview over the process in contrast to the postbox method. We recommend this method if the subbranches are relatively long, so that the effort of generating more than one checklist is somehow justified.

The mentioned execution methods and the corresponding checklist designs are only some suggestions; clearly many other versions are imaginable and of course different versions can be mixed, as we would probably suggest in the situation of Fig. 8, visualised with pseudo-BPMN modelling tools. Here, the two long subprocesses $l1$ and $l2$ can be executed with two separate checklists whereas the short subprocess is included into the checklist in a (dynamic) sequential way, i.e. it would be performed before or after the two long subprocesses. For only one comparatively longer subprocess, a separate Process Checklist for this subprocess is probably not needed as timesaving effects are not that significant. However, this depends on the respective process.

4.5.3 Execution of Inclusive Splits

It is also possible that several subprocesses can be executed in parallel, but at modelling time it is not clear how many of them need to be executed. Unlike for exclusive splits, certain conditions, that is more than one, may be fulfilled and there may be more than one subprocess executed. If this is the case, a control point marked with the word

OR and the condition/question in field *CO* is included in the checklist. Possible answers and a finally-go-to number are listed in field *GT*. So, the resulting control point has elements of control points resulting from both exclusive and parallel splits. The inclusive split, sometimes also called multi-choice split, may be included in the checklist like the dynamic sequential execution of parallel splits, i.e. the go-to numbers after each answer in field *GT* refer to other checklist points (without backward jumps!) or it may be included in the checklist like the parallel execution method, i.e. the go-to numbers in *GT* refer to subchecklists. Depending on the modelling method, additional control nodes referring back to the inclusive decision control node, as is the case for the sequential dynamic method, are needed. All subprocesses to be executed have to be marked in the box □ in field GT, at least one box up to all boxes. For the marked boxes and the corresponding subprocesses, it is executed like for parallel splits. Again, variable c is set to $c = 1$ as the inclusive decision control point may be processed more than once. Of course, inclusive splits may also be executed with the postbox method, taking over the checklist design of the dynamic sequential method, but in contrast to parallel splits, the number of executed subprocesses may vary from process instance to process instance and the involved agents have to pay even more attention to which activities need to be executed and which not.

When using parallel or inclusive operating points with the postbox or the multiple-checklist method, special attention has to be paid to the documents so that one document is not needed in two subprocesses at the same time (if there is no copy available). For the postbox method, this does not necessarily lead to deadlocks, but it increases processing time if agents want to proceed a task but not all needed documents are available at this point of time. The postbox method fails when a receipt system is established, as the checklist and particularly the documents are not handed over but are deposited more or less anonymously in a box. The checklist designer, for example the modelling expert and/or an executing agent who is familiar with the process (see Sect. 6 for the tasks of the designer), has to weigh which execution method is best for each parallel or inclusive split situation.

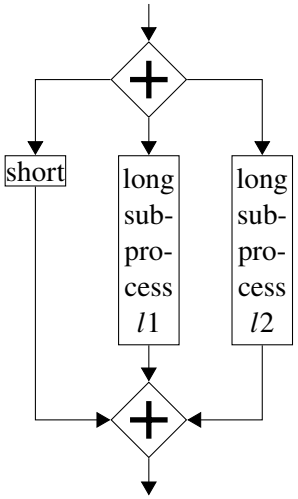

Figure 8: Schematic part of a process model with one short and two comparatively long parallel subprocesses (i.e. consisting of a lot of tasks) that can be executed as a composition of presented methods.

5 Transformation of Process Model Elements through Serialisation

This section focuses on generating a checklist out of an existing BPMN process model. This means in general that a multi-path structure is translated into a sequential structure, which is induced by the Process Checklist through the checklist point numbers. The reason for this transformation is the following: (Graphical) Process modelling languages are well-known and can be checked for consistency whereas modelling directly with the Process Checklist is not so easy. Issues like deadlocks or requiring the same document at the same time at two different places have to be avoided. Thus, a proper (graphical) process model, which also may be discussed among the participants and modelling experts, can be derived first and afterwards the Process Checklist is generated out of the model. This transformation process does not take a long time, in contrast to the modelling time itself, and the additional expenditure is therefore small. The graphical process model can later

on also be used as basis for the implementation of an IT-based WfMS. In this context, the Process Checklist can also serve as an instrument for checking the content correctness of the modelled process without the need to implement this process in an IT-based WfMS. Corrections and discrepancies can be recorded on the checklist immediately during the execution. It may also be the case that process models are already available, e.g. for documentation purposes, and so the transformation is the only thing that has to be done to get a feasible support tool. BPMN should thereby just be seen as an illustrative example for graphical process model languages.

In the following, it is explained in which way the single elements of the (BPMN) process model are transformed into either operating points or control points, i.e. are serialised. Instead of transforming a BPMN process model directly into a form suitable for a WfMS, the model is converted to a checklist vector, which is the basis for the Process Checklist. This transformation process is indicated in Fig. 9 with the double-lined arrow.

The problem of transforming a model drawn in one business process modelling notation into another notation has been examined in different papers, e.g. by Hauser et al. (2006) and Koehler et al. (2008). However, to the best of the authors' knowledge, the transformation of process models to a checklist representation has not been discussed so far, except for in Baumann et al. (2014), which is the basis for the work at hand.

The transformation steps are performed in an algorithmic way, except for parallel and inclusive gateways. There, the modeller has to decide which of the execution methods presented in Sect. 4.5 is best in each situation. But first of all, before specifying the transformation of process models into checklists, we have to determine how suitable process models should look. These specifications are necessary to give concrete mapping rules. For process models, only basic elements of BPMN are allowed. As zur Muehlen and Recker (2008) show, this is enough in most cases. Also, the paper at hand has to be seen as a first step in this topic.

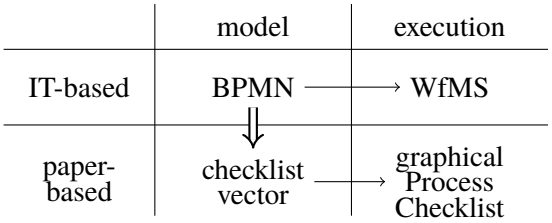

Figure 9: Schematic approach to differentiate paper-based from IT-based process management systems.

A process model for which we want to demonstrate the transformation into a Process Checklist is defined according to BPMN 2.0 (see, e.g. Object Management Group Inc. 2011) allowing for the following basic elements:

- flow objects: activities, events (i.a. start and end event), gateways (AND, XOR, OR)
- sequence flows
- data (input/output) objects
- participants: one pool, possibly separated into different lanes

One requirement that we impose and that is only a representational one is that each split gateway has a corresponding join gateway. This requirement simplifies the following explanations a lot. The process model should be sound in a sense that it can be successfully executed without containing deadlocks (a situation where two or more subprocesses are blocking each other), livelocks (a situation where an endless repetition of one or more activities happens) or dead activities (activities that can never be reached). If the model is sound, then the checklist derived from the model is sound as well because the model and the checklist exhibit the same behaviour, i.e. we can always transform a sound process model to a Process Checklist. More precisely, it depends on the modeller whether all behaviour of the process model is reproducable with the checklist. The other way round it is always true: The complete checklist behaviour can be reproduced by the original process model. With same behaviour we mean that the same execution paths are possible when

identifying the activities in the model with their corresponding operating points in the checklist. Nevertheless, exceptions may occur during the execution of a Process Checklist based on a sound process model (which might end in an unscheduled or non-successful execution), see Sect. 4.3.

As we consider the application of checklists appropriate only within one company, there should not occur processes with more than one pool. Therefore, we do not have to take message flows into account. Although it is possible to map message flows to a Process Checklist as done in one case study shown in Sect. 7.2 and Fig. 24, we simply have not specified a standardised representation of message flows yet. At the moment, message flows can only be mapped in a generic way through operating points containing instructions like "Wait for an answer". Furthermore, message flows are not part of all process languages and also not one of the basic elements of BPMN (zur Muehlen and Recker 2008).

Further on, this section has to be seen as an example of how to transform graphical model elements. Extensions surpassing the presented mapping instructions can easily be included at any time. Which specific forms of activities, events and gateways can be covered with the transformation rules for these kinds of models will become apparent when it comes to the concrete transformation of process models into checklists.

5.1 Transformation of Activities

Activities are transformed straight into operating points p^o. Their descriptions are mapped on the field AC whereas all directly incoming data and directly outgoing data are mapped on the field ID and OD respectively. The participant of the corresponding lane or hierarchy of lanes, which may be a single person or a role specification, is mapped onto the field AG. Concrete agent names may be filled in during the execution of the checklist. An example of an activity with documents and participants is given in Fig. 10 and the corresponding checklist point, an operating point, in Fig. 11. We should mention that AG_i is a dummy for the agent/role of point i and not a certain agent's identifier.

Figures 10 and 11 show an excerpt from an abstract process model with labels according to an operating point p_i^o. It is set $ID_i = (IDO_1, IDO_2, IDO_3)$ and $OD_i = (ODO_1, ODO_2)$ where IDO_k and ODO_l, $k = 1, 2, 3$, $l = 1, 2$ are identifiers for single data objects.

5.2 Transformation of Subprocesses

Occurring subprocesses, marked with a symbol as seen in Fig. 12, may be taken into a checklist in different ways:

1. Include the complete subprocess. This leads to a comparatively long but easy to understand checklist.
2. Generate a new checklist for each subprocess. One control point j has to be inserted into the original checklist with work instructions for printing and passing on the new checklist ($a_{j,1}$ ="print and pass new checklist named SCL to agent/role Y, go to SCL:", $g_{j,1} = 1$ (of the subchecklist)), which has to be confirmed due to $c_j = 1$, and with instructions for waiting for this checklist to come back completely processed. Agent/role Y is the agent/role description of the first point in the subchecklist. Parameter $a_{j,2}$ is set to "finally go to" and $g_{j,2} = j + 1$ (of the main checklist). (This is not needed here when simply regarding the sequence flow.) This is according to the multiple checklist method for parallel splits presented in Sect. 4.5.2, Paragraph *Parallel Execution*. An example for a control point resulting from this method of transforming subprocesses is shown in Fig. 13.

A selection between those two methods could be effected considering the length of the subprocess behind the BPMN subprocess task. Short subprocesses may be directly included into the main checklist whereas long subprocesses should be put into a subchecklist to maintain better clarity.

Subchecklists are recommended especially in cases where subprocesses contain backward

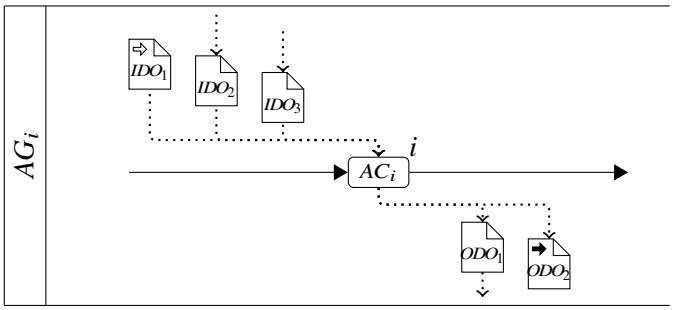

Figure 10: Abstract BPMN representation of an activity with ingoing and outcoming documents.

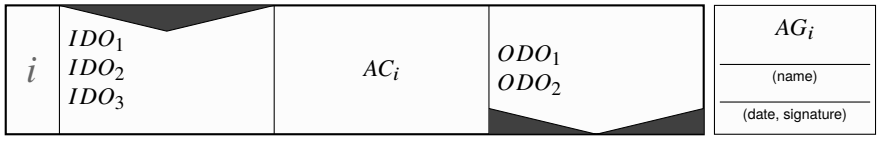

Figure 11: Representation of the activity of Fig. 10 as an operating point of a checklist.

jumps, i.e. the loop is completely within one subbranch. When the loop is executed, the whole main checklist does not have to be reprinted, only the subchecklist of the subprocess, which is presumably shorter.

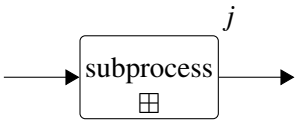

Figure 12: Symbol for a subprocess in BPMN 2.0.

5.3 Transformation of Gateways

As already the design and execution of splits and joins of a directly generated checklist imposes several possibilities, so does the transformation of BPMN gateways to checklist elements. Most of the methods described in the following correspond to one execution method presented in Sect. 4.

5.3.1 Transformation of Exclusive Gateways

An exclusive split gateway (see Fig. 14) has to be transformed into a control point in which the decision question (CO_j) and the possible answers ($a_{j,1}, a_{j,2}$ and $a_{j,3}$ in Fig. 14) with the respective go-to numbers ($g_{j,1} = j_1, g_{j,2} = j_2$ and $g_{j,3} = j_3$ in Fig. 14) are mentioned. Parameter c is set to $c_j = 0$ as the decision has not to be confirmed in field GT_j. Also, it is set $a_{j,4}$ = "" and $g_{j,4}$ = "" because no line without □ box is needed. The control point is executed only once.

For the further transformation, two cases have to be considered: Does the split gateway induce a backward jump (loop) or are all alternative subbranches pointing into the future? If there is an exclusive join gateway (Fig. 15) after the split gateway, that means two or more branches point into the future, a jump instruction to the next point in the checklist after the join gateway (j_4 in Fig. 15) must be inserted at the end of each branch in the checklist, see Sect. 4.5.1. Only the branch listed last in the checklist does not need this jump instruction as the point after the exclusive join is performed next in any case. Jump instructions also do not need a □ box line in GT.

When arrows coming out from exclusive split gateways point back into the past, then consecutive checklist numbers need to be introduced and the checklist point numbers on the cover sheet have to consider the checklist numbers, too (see Fig. 6). A join gateway (exclusive join) catching the backward jump does not need to be considered in the transformation, i.e. it is not represented separately. When there is a jump out of a parallel or inclusive subprocess, which is not the soundest way of

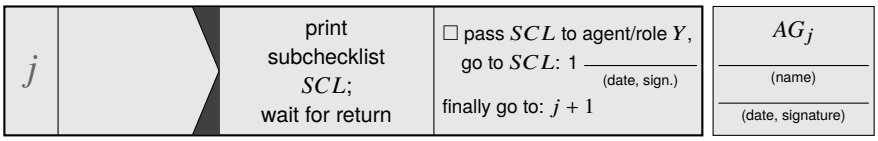

Figure 13: Representation of the subprocess task of Fig. 12 according to the second method (distributing a subchecklist) as a control point of a checklist.

modelling, then an operating point to inform the agent responsible for synchronising the parallel or inclusive subprocesses about the interruption of the parallel or inclusive subprocesses has to be included into the checklist. That means, legwork by hand is necessary in such a case. Otherwise this agent would wait for all subprocesses to be finished, which will not happen. An exemplary control point for an exclusive split without a backward jump and one jump instruction (performed by agent AG_{j+n}, which is usually the same as agent AG_{j+n-1}, i.e. the agent of the last task in the subbranch) is given in Fig. 16.

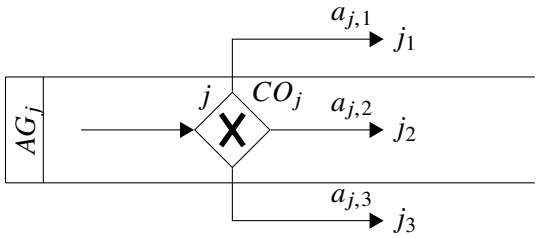

Figure 14: Exclusive split gateway without a backward jump.

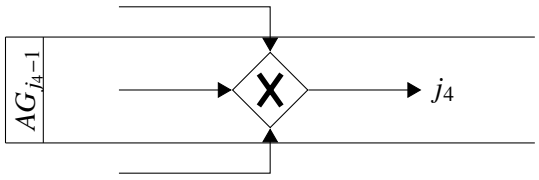

Figure 15: Exclusive join gateway (without backward jump) that does not have to exist if the outgoing branches of the exclusive split gateway end with terminal events.

5.3.2 Transformation of Parallel Gateways

The transformation methods of parallel gateways correspond to the ones already listed in Sect. 4.5.2 about the execution of parallel splits, except for the one named *static sequential transformation*, which is explained in the following. Advantages and disadvantages of the different possibilities were already mentioned in Sect. 4. Of course, a mixture of several transformation methods is possible, too (see, e.g. Fig. 8 and its explanation).

Static Sequential Transformation

This type of transforming a parallel gateway takes the several branches of the process model that are between the split and join gateway and brings them into an arbitrary order or an order that seems to be reasonable at transformation time. The resulting checklist part is purely sequential. The gateway itself is not mapped to the checklist. This transformation method keeps the checklist very simple, but execution time may increase as flexibility allowed by parallelism is completely ignored.

Dynamic Sequential and Postbox Transformation

This transformation results in a checklist as described in Sect. 4.5.2, Paragraphs *Dynamic Sequential Execution* and *Postbox Method Execution*. The parallel split gateway will be transformed into a control point p_j^c. An exemplary transformation of a parallel split as shown in Fig. 17 into a control point is demonstrated in Fig. 18. The parallel subbranches of the process model have to be written down sequentially in the checklist. At the end of each branch, a jump to p_j^c, realised with a simple control point like in Fig. 16 with description "AND end", except that the go-to instruction points back to point j instead of j_4, is necessary. Furthermore, in p_j^c the number of the point following the respective parallel join, in Fig. 17 that one that gets number j_4 in the checklist, has to be noted (the finally-go-to part in Fig. 18). The

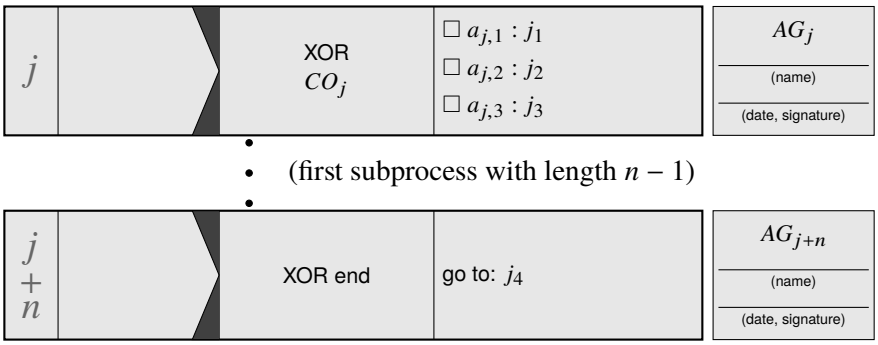

Figure 16: Representation of the exclusive gateway of Fig. 14 and a jump instruction after the first subprocess transferring the performer to the task with checklist number j_4 after the join gateway of Fig. 15. Usually, when the process model is transformed successively, it is $j_1 = j + 1$.

parameter specifications are in detail: AN_j = "", CO_j = "AND", $c_j = 1$, $a_{j,1}, \ldots, a_{j,3}$ = "go to", $g_{j,1} = j_1$, $g_{j,2} = j_2$, $g_{j,3} = j_3$, $a_{j,4}$ = "Finally go to", $g_{j,4} = j_4$. For the jump instructions (three are needed in the example of Fig. 17 as there are three branches), the parameters are set as follows: AN_{j_k-1} = "", CO_{j_k-1} = "AND end", $c_{j_k-1} = 0$, $a_{j_k-1,1}$ = "go to", $g_{j_k-1,1} = j$, $k = 2, 3, 4$.

Parallel Transformation

For each parallel subbranch, a checklist is generated and distributed by the agent of the split gateway (AG_j in Fig. 17) to the agents of the first process elements of the subbranches. As introduced in Sect. 4.5.2, Paragraph *Parallel Execution*, it is modelled as one control node p_j^c. If the gateway splits into k branches, then $a_{j,k+1}$ = "finally go to" and $g_{j,k+1} = j+1$. If the name of the current checklist is "Checklist", then CO_j = "AND—print checklists Checklist_sub1, ..., Checklist_sub$k$", if the names of the subchecklists are 'Checklist_sub1', ...,'Checklist_sub$k$'. Of course, $a_{j,1}, \ldots, a_{j,k}$ have to reference these subchecklists, $g_{j,1}, \ldots, g_{j,k} = 1$ (i.e. the first points of the subchecklists) and $c_j = 1$, which means that signatures for all returning subchecklists are needed (see Sect. 4.5.2 and Sect. 5.2). At the end of each subchecklist, a control point referring back to the subchecklist distribution point of the main checklist has to be inserted.

5.3.3 Transformation of Inclusive Gateways

The transformation of inclusive gateways can be done similarly to the transformation of parallel gateways. More precisely, there are the possibilities to use the dynamic sequential or postbox transformation or the parallel transformation (see also the several execution possibilities in Sect. 4.5.3). The only difference is that in p_j^c we have CO_j and $a_{j,1}, \ldots, a_{j,k}$ as in the exclusive gateway transformation, i.e. the condition/question and the answers have to be taken over from the process model. An 'else' case, if present in the BPMN model, can simply be transformed by $a_{i,k}$ = "else" and $g_{i,k}$ pointing to the respective checklist point.

5.4 Transformation of Events

The transformation of BPMN events is dealt with very briefly in this paper, as we suggest ad-hoc transformations of events involving good capabilities of the checklist designer. This can be the modelling expert and/or an executing agent who is familiar with the process. It is one time individually decided whether to include an extra operating point for the event or to include its condition in the following checklist point. Like the multitude of activity types, there are a lot of different event types. Providing transformation rules for all of these types would not be expedient in our opinion, as the work at hand is supposed to give a basic overview of Process Checklists, their structure, usage, advantages and disadvantages.

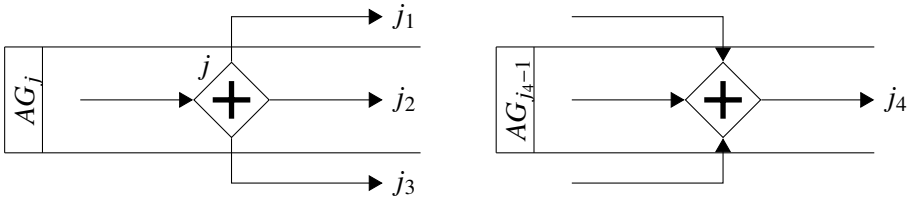

Figure 17: Parallel split gateway p_j^c and parallel join gateway $p_{j_4-1}^c$.

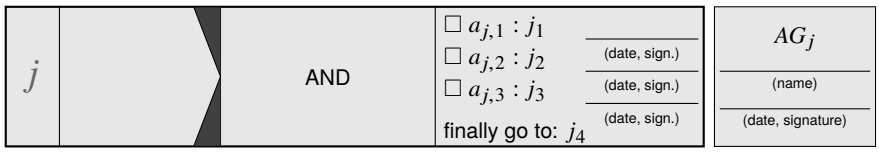

Figure 18: Checklist representation of the parallel split in the dynamic sequential way without the corresponding jump instructions that refer to agent AG_j of point No. j after every subbranch.

5.4.1 Direct Transformation of Events

Some events, like signal, escalation or compensation events, as well as milestones (untyped catching intermediate events) can be transformed like activities, i.e. to operating points p_i^o, where AC_i is used for transmitting some message (e.g. an escalation instruction) or AC_i = "" (e.g. for letting a supervisor know that the process has reached a certain stage).

5.4.2 Indirect Transformation of Events

Certain events, like time, condition and message events, represent some requirements for the next point in the checklist and can be modelled this way. An example for a transformed catching timer event is given in Fig. 19. The requirement is written down in *AC* or *AN* of the following operating or control point.

5.4.3 Ignored Events

Other events, like the start event, can be ignored, which means they have no representation in the checklist, because they won't influence the execution. Usually, the processing of the checklist is started by the process owner and ended by returning the checklist to the process owner. Therefore, start and end event are executed automatically through printing the checklist and returning it when finished.

5.5 Transformation and Execution of Infrequent, Mutually Exclusive Activities or Branches

As a paper-based checklist needs direct human treating during its execution (see Sect. 4), it can be handled very flexibly by the agents. This does not mean that the agents can do what they want during the execution. Rather they have a certain freedom, which is not unrestrictedly given when processing with, e.g. the aid of a WfMS. Consider a clinical workflow where a diagnosis (outgoing data) has to be made in one step and, according to this diagnosis, the treatment has to be executed in the following step (diagnosis as incoming data). Implementing all different kinds of diagnosis-depending treatments would cause an exclusive gateway with nearly innumerable branches or subprocesses, not to mention that all these eventualities have to be considered at modelling time (cf. Ely et al. 2011). What if a certain treatment has been forgotten because of its rarenes or if it was yet unknown at modelling time, for example?

When facing this problem in the context of checklists, the following solution is conceivable: List only the most frequently made diagnoses in the corresponding XOR-control point $((a_{j,1}, g_{j,1}), \ldots, (a_{j,l}, g_{j,l}))$ and add one $(a_{j,l+1}, g_{j,l+1})$ with $a_{j,l+1}$ = "other" and $g_{j,l+1}$

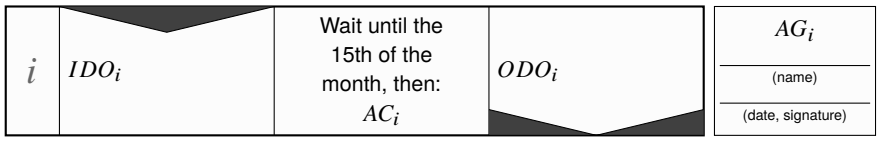

Figure 19: Indirect representation of an intermediate catching timer event in the event-following activity number i.

referring to an empty operating point where the concrete diagnosis and all incoming and outgoing data can be entered at running time by the doctor in charge. These empty operating points offer a way to reduce complexity of the process model and to prevent the process from getting stuck during its execution. But as they require good knowledge about the process, they can only be filled in by agents with the corresponding expertise and should therefore not be overused.

It should be mentioned, as briefly indicated above, that also in traditional WfMSs flexibility can be achieved. In the literature, when talking about flexibility, it is distinguished between different types of flexibility: flexibility by deviation, flexibility by underspecification or flexibility by change (see Mans et al. 2009). However, as stated in Sect. 1 and, for example, by Pešić (2008) and Zeising et al. (2014), WfMSs usually do not support all of the different types of flexibility due to their conceptional basis. When considering checklists, the required flexibility can be achieved in a fast and easy way. A more detailed flexibility discussion of checklists is given in Sect. 7.1. To conclude this section, an overview of common process patterns taken from van der Aalst et al. (2003) that the Process Checklist is (not) able to reproduce is given in Tab. 1 and 2. For a detailed description of the patterns see van der Aalst et al. (2003).

6 Implementation

Although process execution of the Process Checklist approach is mainly based on paper, it requires some implementation. Figure 20 depicts the general flow of work in the Process Checklist approach. We assume that in a first step, a formally correct BPMN process model is specified. Therefore, any process modelling system is qualified that is able to export that BPMN process model as an XML data structure. This XML model is input for the model transformation system called PCL Generator that we have to deliver. The PCL Generator takes the XML process model and transforms it into a LaTeX file (see Sect. 6.2). The latter can be imported by a LaTeX processor in order to create the Process Checklist as pdf file. As an additional service, the PCL Generator also produces an XML model of the Process Checklist what supports sharing and exchanging a Process Checklist (see Sect. 6.3).

In the following, we describe the implementation of the Process Checklist approach, which is a stand-alone implementation, give insights in the underlying meta-model as well as the model transformation procedures. Furthermore, we describe the XML-based serialisation method of checklists with an example.

6.1 The Checklist Meta-Model

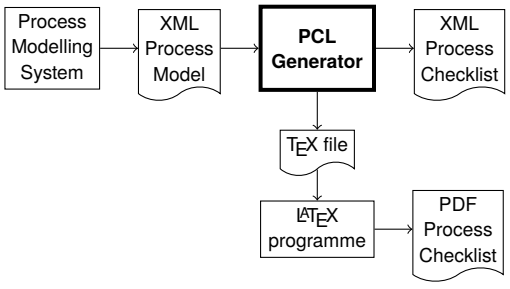

Figure 20: Illustration of the flow of work.

The checklist meta-model as a UML class diagram is shown in Fig. 21 where all elements introduced in Definition 1 can be found again. The meta-model describes the general representation structure of checklists and serves as an implementation template. When a new checklist is created manually or a BPMN model is transformed to a

	pattern	checklist solution
basic control flow patterns	sequence	numbering of the checklist (operating) points
	parallel split	control points (see Sect. 4.5.2, 5.3.2)
	synchronisation	control points (realised in the same control point as parallel split plus redirection control points)
	exclusive choice	control points (see Sect. 4.5.1, 5.3.1)
	simple merge	control points (for redirection instructions)
advanced branching and synchronisation patterns	multi-choice	control points (see Sect. 4.5.3, 5.3.3)
	synchronising merge	control points (realised in the same control point as multi-choice plus redirection control points)
	multi-merge	not explicitly described in this paper; multi-merge can be modelled alternatively (see van der Aalst et al. 2003) as synchronisation
	discriminator	not described
structural patterns	arbitrary cycles	control points (see backward jump in Sect. 4.5.1, 5.3.1)
	implicit termination	not explicitly described in this paper; can be realised through a checklist point that has no go-to instruction and is not followed by another point or through a 'terminate' note; this only makes sense if the subprocess is executed with a subchecklist (otherwise there would be a deadlock)
state-based patterns	deferred choice	can be realised the same way as exclusive choice without decision/question; however, the agent has to choose between alternative subbranches when he reaches the control point
	interleaved parallel routing	control points of dynamic sequential type (see Sect. 4.5.2, 5.3.2)
	milestone	operating point (see Sect. 5.4)

Table 1: Common process patterns concerning process design and their representation in the Process Checklist (part 1)

	pattern	checklist solution
patterns involving multiple instances	multiple instances without synchonisation	distribution of several checklists (without control point in a main checklist to collect them); consistent numbering of the checklists required so that they can be assigned to one instance
	multiple instances with a priori design time knowledge	distribution of several checklists (runtime) or repeating the activity in the checklist as separate operating points (design time)
	multiple instances with a priori runtime knowledge	distribution of several checklists (runtime) or, if maximum number of instances given, repeating the activity in the checklist as separate operating points with go-to instructions to leave the repetition (design time)
	multiple instances without a priori runtime knowledge	distribution of several checklists
cancellation patterns	cancel activity	not easily realisable; information about cancellation has to be brought to the agent at the right time
	cancel case	can be carried out by the process owner at his periodical milestone checkings (not immediately; same problems as with cancel activity)

Table 2: Common process patterns concerning process execution and their representation in the Process Checklist (part 2)

checklist, a new instance of the meta-model is created. A checklist consists of several *Checklist Elements*, which can either be *Operating Points* or *Control Points*. An OperatingPoint can have several ingoing as well as outgoing *Data Objects* and is performed by exactly one *Responsible Role*. A ControlPoint has several *GoTo* instructions and has also exactly one ResponsibleRole. A GoTo instruction refers to a Checklist Element again. The confirmation attribute of ControlPoint corresponds to variable c in the definition of checklist vectors (Definition 1), which specifies if a confirmation line for signing is added in field *GT* behind the go-to numbers of control points (e.g.for parallel and inclusive splits, confirmation lines are added).

6.2 Model Transformation and Checklist Generation

In order to provide a simple transformation possibility, the described transformation procedures from BPMN to a Process Checklist representation have been implemented in a C-Sharp application using the Microsoft .NET framework. In this way, Process Checklists can be generated from existing BPMN models without great effort, depending on the size of the original model.

In a first step, the user selects the BPMN-XML file that should be transformed from a file dialog. Subsequently, the selected file is parsed by the application. As a result, an instantiation of the (simplified) BPMN meta-model is generated. Note that for this approach we are only considering the basic BPMN elements as described in Sect. 5. Afterwards, the user has to choose the transformation method of possibly occurring parallel gateways, i.e. whether a static sequential, a dynamic sequential or a parallel transformation method is preferred. The provided information is finally used to transform the BPMN model instance to an instance of the checklist meta-model

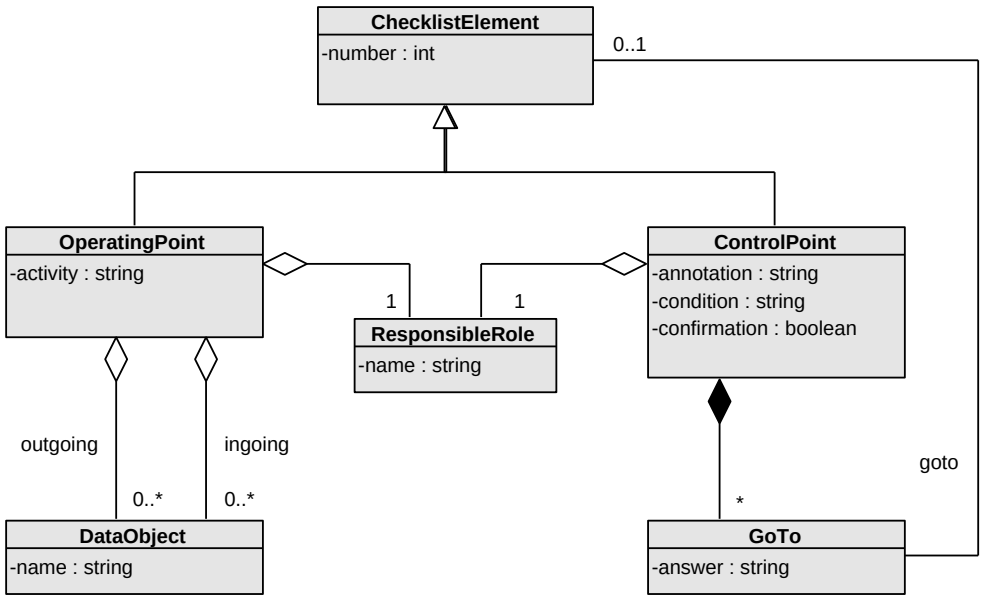

Figure 21: Meta-model of the Process Checklist.

as shown in Fig. 21. Adjustments or modifications such as the milestones mentioned in Sect. 1 for the checklist owner's periodical revision can be added afterwards to the instantiated meta-model. Generation time then strongly depends on the number of such manual modifications.

The different checklist model elements can finally be read and visualised. Of course, there is no fixed checklist visualisation. However, the visualisation method presented in the work at hand is the result of thorough discussions with process participants and modelling experts.

In our implementation, we are generating graphical checklists by means of LaTeX files using the *tikz*-package to draw diagrams. Accordingly, the checklist model needs to be transformed to LaTeX-code. We defined two new LaTeX commands CP for control points and OP for operating points, which are called with different parameters. Shapes, fonts and colors are defined within these command definitions and can therefore be adjusted to the users' needs. An operating point *perform exam correction* with the running number 5, which consumes a data object *exam unmarked*, produces a data object *exam marked* and should be performed by the person (role) *auditor*, is presented by the following LaTeX-code:

```
\OP{5}{exam unmarked}{perform exam
correction}{exam marked}{auditor}{1}
```

The last entry '1' is for adjusting the graphical representation. Based on LaTeX, the graphical checklist generation is independent of the system environment. Furthermore, representational details can be adjusted directly by users and independent of a specific implementation. The graphical checklist can then be saved and printed as LaTeX produces a pdf file.

6.3 Checklist Serialisation

In order to be able to adequately save and share generated checklists, additionally, a possibility to serialise checklists to XML[1] is provided. The listing in Fig. 22 shows an excerpt of the generated XML code of the example checklist vector of Fig. 5.

[1] Complete example checklist XML files as well as an XML schema definition can be found at the project website. See checklists.kppq.de or ai4.uni-bayreuth.de/de/research/projects/002_processchecklists for more information.

```
<checklist process='Administering an exam'>
      ...
      <controlpoint no='2' annotation=} condition='Exam type?'
       confirmation='0'>
            <gotolist>
                  <goto answer='written' gotoNo='3' />
                  <goto answer='oral' gotoNo='9' />
                  <goto answer=} gotoNo=} />
            </gotolist>
            <role name='Student' />
      </controlpoint>
      ...
      <operatingpoint no='5' activity='Perform exam correction'>
            <ingoing>
                  <DataObject name='Exam unmarked' />
            </ingoing>
            <outgoing>
                  <DataObject name='Exam marked' />
            </outgoing>
            <role name='Auditor' />
      </operatingpoint>
      ...
</checklist>
```

Figure 22: XML code sample for checklist serialisation

Out of the data stream as in Fig. 22, a checklist based on the meta-model of Fig. 21 can be reconstructed independently of the underlying architecture.

7 Evaluation and Analysis

In this section, we evaluate our idea with respect to desired features and report on two case studies, one in the academic field and one within a bank, *Sparkasse Bamberg*. In these case studies, Process Checklists are examined with regard to several criteria. One of them is flexibility. As there exist several interpretations of flexibility, we provide a short discussion about different flexibility types in Sect. 7.1. The other criteria, namely parallelism, length, comprehensibility, orientation and reliability, are self-descriptive. Sect. 7.2 and 7.3 represent the case studies.

Concerning effectiveness we had good experiences within the two case studies as all of the executed processes were completely and correctly finished. Problems like forgotten activities did not occur. Processing times were appropriate, compared to having no support, although the checklist was just introduced during the case studies. We expect to lower processing times when process participants get more familiar with the checklist, but we also do not expect to reach the processing times of IT-based execution support in general. In case of electricity failures or other technical problems, the checklist can still be used.

For evaluating the desired advantages of paper-based checklists as well as showing potentials for improvement, the two case studies were carried out in different areas of application. The first case study was performed in the academic domain. The second one in the financial business area. In the first case study, we primarily want to verify the power, i.e. the practicability of the model requirements as well as the comprehensibility by the user. The second case study addresses the user's comprehensibility and the acceptance of the user again, which is an important factor when introducing new procedures in organisations.

Additionally, we want to get information whether Process Checklists are needed at all.

Primarily, the first evaluation is a comparison of process execution through checklists and through BPMN models and a comparison of the different transformation methods shown in Sect. 5 or, respectively, of execution possibilities shown in Sect. 4. Basically, there are different forms of deploying graphical BPMN models. A very simple form is to publish BPMN models, for example on the intranet of a company, as hard copies or on a black-board. Process participants now are obliged to take these models into account when working on processes. Although this enactment is very cheap, it lacks accountability. Another form of process execution is to deploy a Process Management System (or WfMS). Although the execution of processes now becomes obligatory, the implementation of a Process Management is very costly regarding the WfMS selection procedure, installation costs, licence costs, costs for teaching end users and building up the IT capabilities to administer the system etc. Many small and medium companies cannot and/or will not afford this form of enactment. Nevertheless, a Process Checklist is a good compromise between these two ways of enactment. A Process Checklist is cheap and it ensures a good degree of accountability. When introducing the checklist, we need a training course for the checklist users, an administrator and the installation of all required documents, e.g. simple pdf documents. In the university where we carried out the first case study, a WfMS was not available but only BPMN models. And also for the second case study, the basis for the evaluation checklist is a graphical process model (modelled in a proprietary process language), which was the only process guidance until then.

The former discussion of pros and cons for process enactment alternatives is the reason why our two project partners chose to use Process Checklists for process enactment.

7.1 Feature-based Evaluation of Checklists wrt. Flexibility

By Schonenberg et al. (2008), four types of flexibility are proposed: flexibility by design, flexibility by underspecification, flexibility by change and flexibility by deviation. Since flexibility is a very important aspect of the deployment of Process Checklists, the latter are evaluated according to this aspect.

'*Flexibility by design* is the ability to incorporate alternative execution paths within a process model at design time allowing the selection of the most appropriate execution path to be made at runtime for each process instance (Schonenberg et al. 2008).' This type of flexibility is achieved by Process Checklists through control points and the different possibilities for designing them as exclusive, inclusive and parallel splits. As Process Checklists can be generated from existing process models, e.g. from BPMN models as described in Sect. 5, they inherit more or less the same degree of flexibility by design as the original process models have.

'*Flexibility by deviation* is the ability for a process instance to deviate at runtime from the execution path prescribed by the original process without altering its process model. The deviation can only encompass changes to the execution sequence of tasks in the process for a specific process instance, it does not allow for changes in the process model or the tasks that it comprises (Schonenberg et al. 2008).' This type of flexibility is fast and easily achieved simply by changing the checklist, e.g. inserting an additional task, deleting an existing one or changing the order of two tasks by swapping the point numbers by hand. These changes only apply for one instance, i.e. for one printed checklist. If there are loops in the process, a hint about the deviation ought be given on the cover sheet.

'*Flexibility by underspecification* is the ability to execute an incomplete process model at run-time, i.e. one that does not contain sufficient information to allow it to be executed to completion. Note that this type of flexibility does

not require the model to be changed at run-time. Instead the model needs to be completed by providing a concrete realisation for the undefined parts (Schonenberg et al. 2008).' In Process Checklists, empty control points can be inserted as standard at the desired places in the model. The concrete realisation is then done by the corresponding agent who fills in all the necessary information like task description and utilised data at run-time. See Sect. 5.5 for an example of empty checklist points.

'*Flexibility by change* is the ability to modify a process model at runtime such that one or all of the currently executing process instances are migrated to a new process model. Unlike the previously mentioned flexibility types, the model constructed at design time is modified and one or more instances need to be transferred from the old to the new model (Schonenberg et al. 2008).' This type of flexibility is the most challenging for Process Checklists. Changing a checklist itself is about as expensive as changing any other process model when requirements like validity, correctness and soundness are considered. Adjusting point numbers and especially go-to numbers is, however, more complex than shifting arrows in graphical process models. But transferring the model changes to running process instances is difficult in that way that it could take some time to locate the checklist at all, even if a (paper-based) receipt system is applied. This is a natural consequence of the paper-based approach that can hardly be fixed without slowing down the processing time of each currently running checklist pass or introducing electronic receipts. Within one institution, the change of a Process Checklist could also be communicated to each employee so that all employees working on a changed checklist can print the new version and continue with the new one as described in Sect. 4.5.1 concerning backward jumps, keeping the old one for reconstruction purposes. This approach, however, needs concrete mapping instructions like 'point number 15 of the old version refers to point number 17 of the new checklist version' for all checklist points so that every employee knows where to proceed with the updated checklist.

In summary, we come to the conclusion that checklists meet three of the four flexibility types proposed by Schonenberg et al. (2008). Flexibility by design is a type that is available in virtually all process modelling languages. Admittedly, this flexibility type is easier to realise for graphical process modelling languages than for checklists. Flexibility by deviation and flexibility by underspecification are feasible quite simply, since changing single process instances just means to 'rewrite' the checklist with a pen. Flexibility by change, however, is a problem and difficult to achieve. Regarding only the flexibility criterion, this is why we suggest using Process Checklists primarily in those fields where modifications of particular process executions are often necessary without permanently changing the underlying standard process, i.e. the checklist template. They are also suitable for situations where the explicit modelling of all alternatives at one decision point is virtually unfeasible. They are not recommended in fields where, for example, the processes strongly depend on normative or legal regulations and where these regulations are changed quite often so that the underlying process has to be changed as well. In these cases, a good flexibility by change would be necessary. An example for such processes would be management standards for quality management or environmental standards.

To get feedback about comprehensibility, acceptance and necessity, the two case studies were carried out and are presented in the remainder of this section. Before continuing with the two case studies, it is relevant to note that even if graphical process models are sometimes the only available process support tools in (mostly small and middle-sised) companies, they are not intended for that purpose. But as Process Checklists address the same target group, the BPMN models in the first case study were used as a reference. Especially for those companies that already have graphically modelled processes, as is the case for the second case study discussed below, the enactment of Process Checklists is inexpensive and could lead to evaluation improvements, such as are shown in the following. We compare purely paper-based execution methods, not execution support with WfMSs.

7.2 Academic Domain Case Study

For evaluating the differences between checklists and BPMN models, we distinguish between objective and subjective criteria. The objective criteria are flexibility, parallelism and length. Which characteristics of the checklists and the BPMN models are used to determine these criteria is stated in Sect. 7.2.1. The subjective criteria, comprehensibility, orientation and reliability, had to be derived from interviews and surveys and an ensuing statistical evaluation. Two different processes were modelled to get more reliable results.

We modelled the example process 'Administering an exam' as a graphical BPMN process (see Fig. 23) and as a Process Checklist, which was derived from the BPMN model according to the method presented in Sect. 6.2. Parts of the checklist are represented in Fig. 1. The corresponding checklist vector, the basis for the graphical checklist, is presented in Fig. 5. The first process of the academic domain case study includes the student, the chair's secretariat, the auditor and the assessor as involved agents.

For the second process, applying for a business trip, again a graphical BPMN process was modelled (see Fig. 24). Additionally, two different checklists to examine the differences presented in Sect. 5.3.2 concerning the transformation of parallel gateways were generated. Involved actors are the head of chair, the chair's secretariat and the applicant. The two different transformations of the parallel gateway appearing in the process model were the static and the dynamic sequential transformation. To do a parallel transformation was not suitable in this context, as the parallel subprocesses were too short to get any substantial advantages of this form. To get results for the subjective criteria, every execution support tool (two BPMN models and three checklists) was evaluated by 21 test persons, which means a total of 105 evaluations. Every test person applied the two or, respectively, three support tools for both processes. We diminished learning effects by changing the order of the support tools (BPMN model and checklist) arbitrarily.

7.2.1 Objective Criteria

First of all, the conditions for the three objective criteria had to be set up. For the criterion of flexibility three possible values are available: A low flexibility value means that during the execution of a process the tasks (including the functional, data, operational and organisational perspective) and their order (the control-flow perspective) as prescribed by the support tool cannot be changed. A medium value means that the order of activities can be customised or that certain elements (e.g. a document or a process step) can be deleted. This refers to a certain degree of flexibility by deviation (see Sect. 7.1). A high flexibility value is assigned, if nearly everything can be adjusted during execution (particularly full flexibility by deviation and flexibility by underspecification, see Sect. 7.1). For example, if the chair's secretary is not accessible, this agent can be changed to another person in all activities where necessary for this single process instance. Flexibility by design is available both in the checklist approach and the BPMN process model, whereas flexibility by change is problematic in both cases. Imagine that the BPMN model is available on several blackboards in an organisation. Indeed, they may be changed with a pen, but it is difficult or nearly impossible to change the model for all instances as well as for the execution of exactly one instance.

The values for the parallelism criterion are distributed in the following manner: A low value is allocated if the execution order is fully determined before process initialisation and is therefore only sequential. A medium value means that the execution order is sequential but determinable at run time to make use of agent availability as stated by Degani and Wiener (1991). The parallelism criterion is rated high if real parallelism is possible, i.e. a fully independent execution of several subprocesses. For the 'Administering an exam' process model, no parallelism values could be distributed as there are no parallel gateways in the BPMN process model.

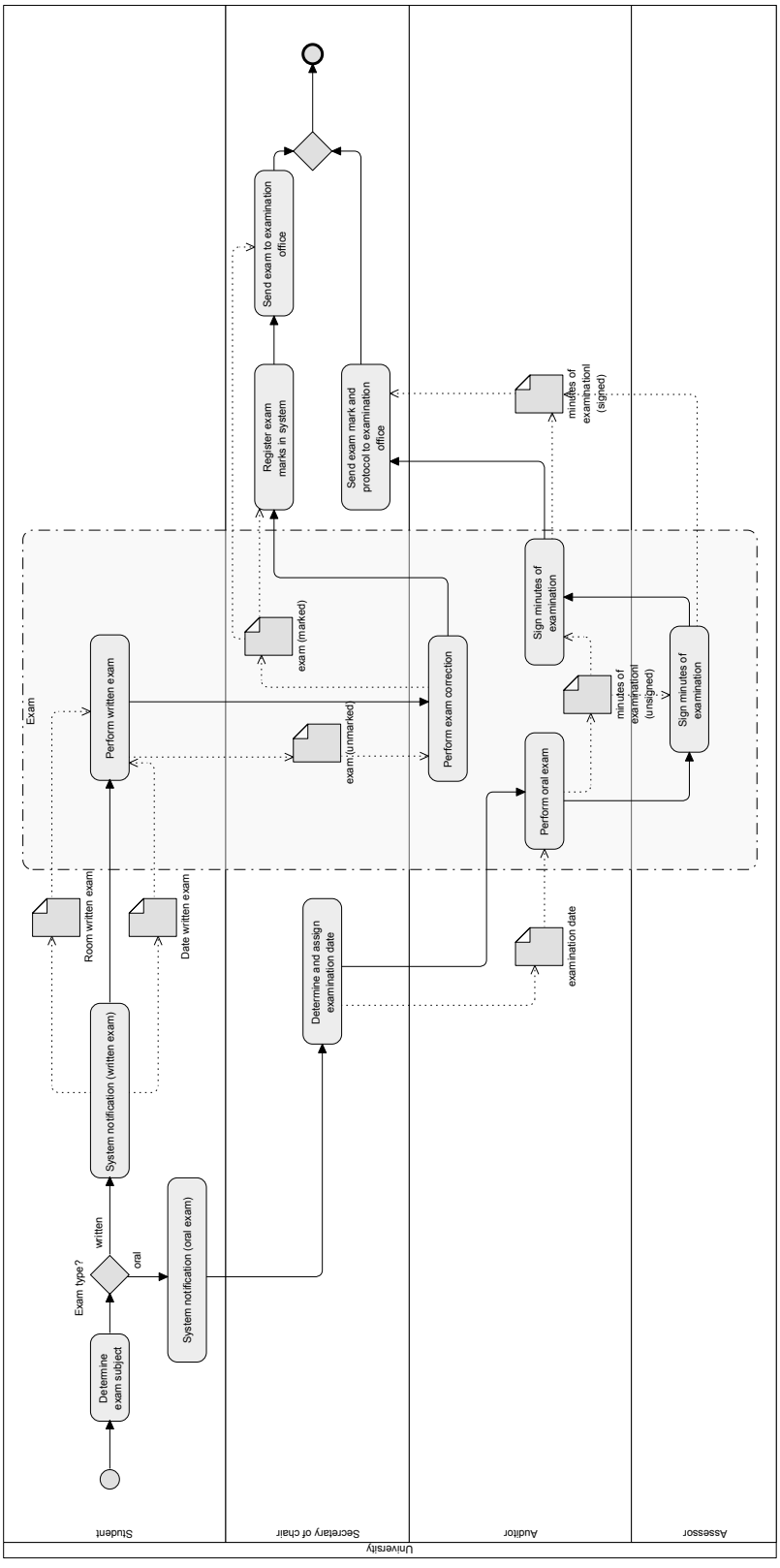

Figure 23: BPMN model for the process 'Administering an exam'.

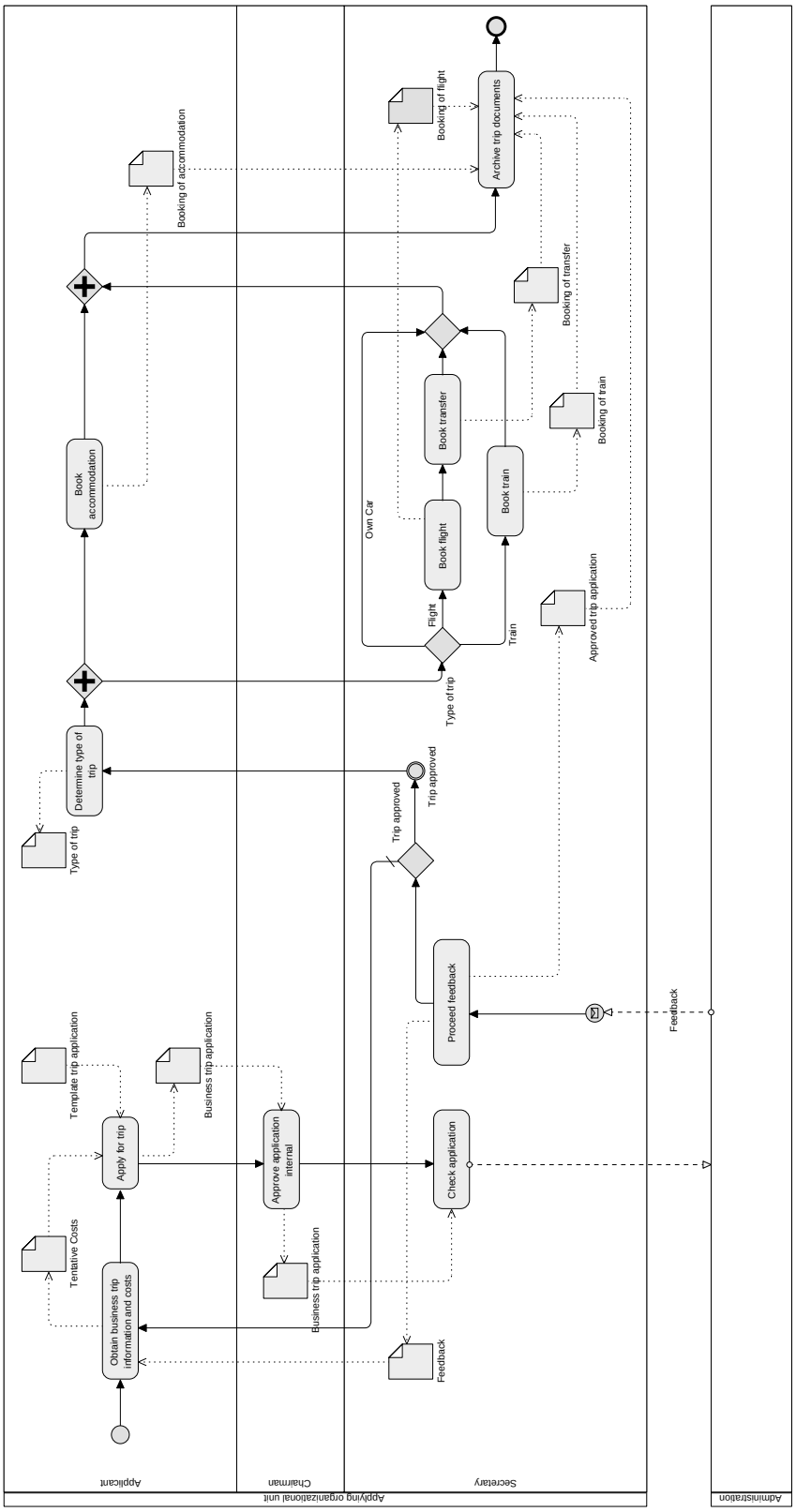

Figure 24: BPMN model for the process 'Applying for a business trip'.

Length of the process execution support tools, i.e. of the checklists and the BPMN models, is just the number of all flow objects in the BPMN model (activities, events, gateways) or the number of operating and control points in the checklists. This value depends strongly on the underlying process and only gives an impression of the compactness or complexity of the different tools and the several transformation methods. Only tools with the same underlying process are comparable with each other. Tab. 3 shows the results of the objective criteria evaluation for the two processes 'Administering an exam' and 'Applying for a business trip'.

7.2.2 Subjective Criteria

As mentioned in the introduction of Sect. 7.2, the subjective evaluation criteria are comprehensibility, orientation and reliability. The 21 test persons—students, PhD students, employees and professors from different chairs and faculties—had to go through the two processes with the help of the different process execution support tools: one BPMN model and one checklist for the first process and one BPMN model and two checklists for the second process. About half of them were not familiar with process models. Afterwards, they rated their impressions of the three subjective criteria by classifying with a Likert scale of five classes. The generated boxplots[2] allow a good interpretation of the results for the three criteria.

In Fig. 25, the boxplots for comprehensibility of the different process execution support tools for both processes are shown. A high value reflects the opinion of the test person that he completely understands the process and the handling of the support tool, in contrast to a low value where the person neither understands the process nor the tool's handling. As it can be seen in Fig. 25, the scattering for both BPMN models is greater than that of the checklists. This could be because some persons—probably those familiar with process models—easily understand the BPMN notation, but others do not, and that the checklists are understood quite well among all test persons. Furthermore, the answers' median for the checklists is higher than that of the corresponding BPMN models. Nevertheless, the shorter but less flexible checklist for the business trip process seems easier to be understood than the longer but more flexible one.

The second subjective evaluation criterion is orientation, which asks about one's own position during the process execution and the steps that have to be performed next. The possible answers ranged from 'When I receive the support tool I know where I am in the process and what I have to do next' to 'When I receive the support tool I neither know where I am in the process nor what I have to do next'. Again, a set of boxplots for the two processes and the two, respectively three, execution support tools was generated and is shown in Fig. 26.

The results for the orientation aspect are similar to that of comprehensibility: Scattering for the BPMN support tools is higher than for the checklists and values for the median are higher, i.e. better, for the checklists than for the BPMN model. As can be seen, the short checklist for the business trip process has a better orientation value than the long checklist. This could be due to the fact that the short checklist provides a slightly better overview over the process and clearer instructions for the tasks. There are fewer control points and thus more longer sequential task chains, consisting of operating points, which allow for easier orientation. Moreover, one conspicuous aspect is that the difference of the BPMN support tools and the checklists is greater for the orientation aspect than for the first aspect, comprehensibility.

In Fig. 27, boxplots for the third subjective evaluation criterion, reliability, are presented. A high level of reliability means that the course of the process execution so far is traceable and it is clear, who has carried out which tasks. A low level means the opposite, i.e. the execution is not traceable and it is not clear who has carried out which tasks. It is noteworthy that the median values for the BPMN support tool are quite low and for the checklists have even increased compared to

[2] For more information about boxplots see Fahrmeir et al. (2016), Sect. 2.2 and The R Foundation (2017).

	Subscribe for exam		Apply for business trip		
	BPMN	Checklist	BPMN	ShortChecklist	LongChecklist
Flexibility	low	high	low	high	high
Parallelism	*NA*	*NA*	high	low	medium
Length	16	15	20	15	18

Table 3: Values for the objective evaluation criteria for the different support tools; NA = not available.

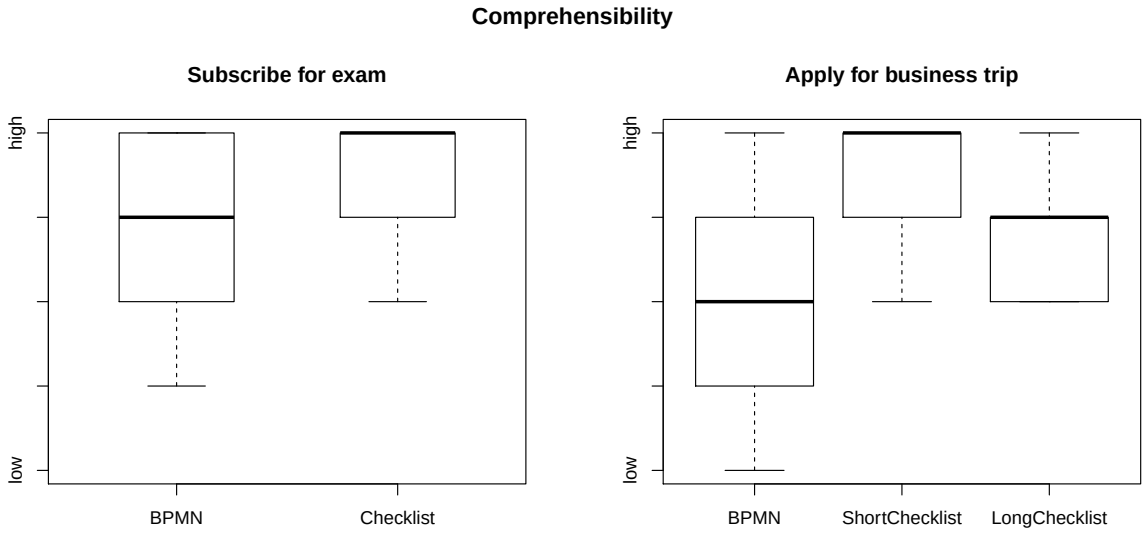

Figure 25: Boxplot evaluation of the comprehensibility for the two processes 'Administering an exam' (BPMN model and checklist) and 'Apply for a business trip' (BPMN model, a short checklist and a long checklist).

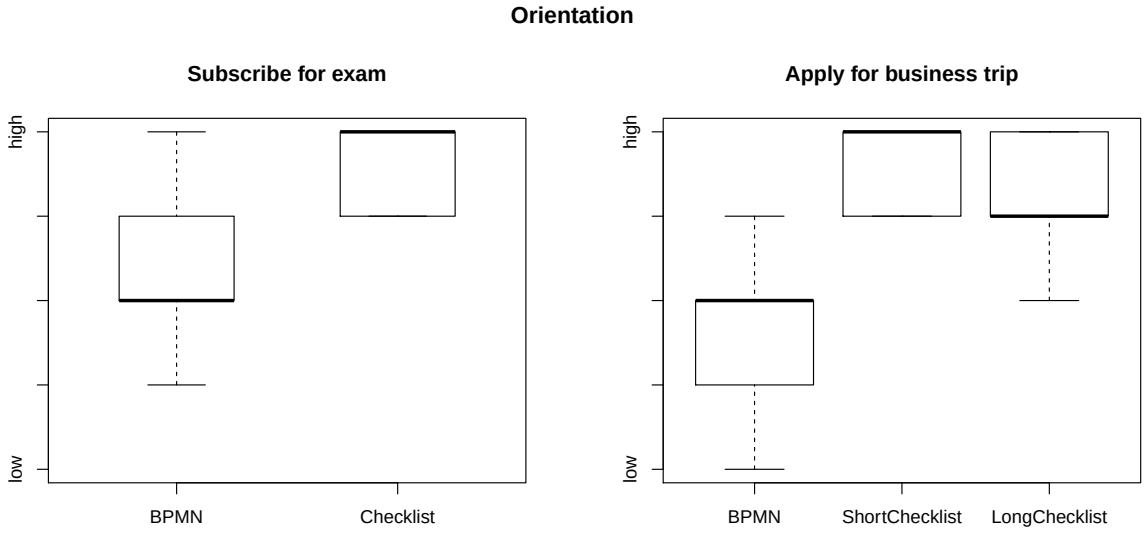

Figure 26: Boxplot evaluation of the orientation for the two processes 'Administering an exam' (BPMN model and checklist) and 'Apply for a business trip' (BPMN model, a short checklist and a long checklist).

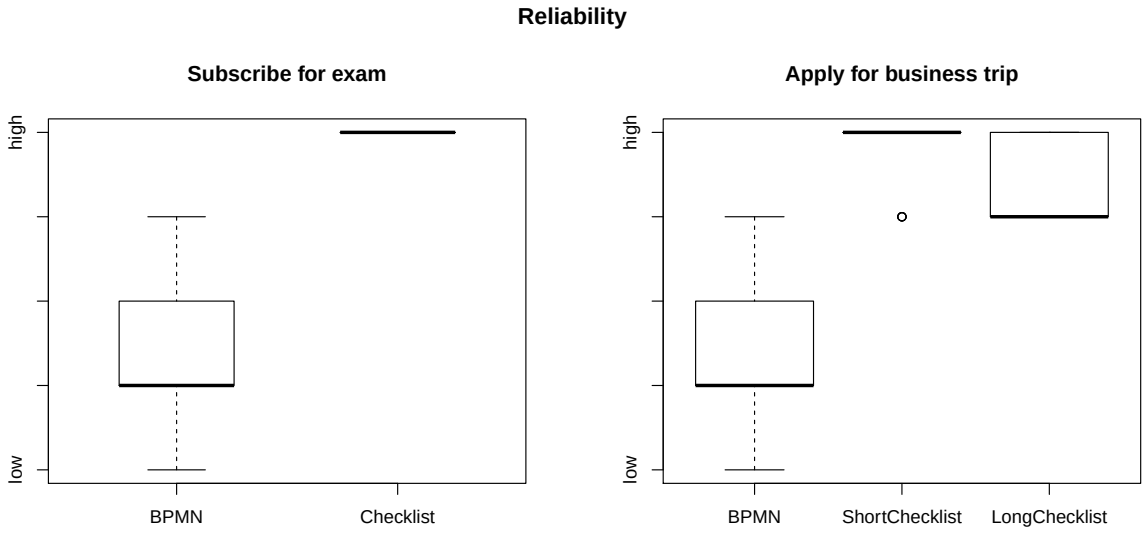

Figure 27: Boxplot evaluation of the reliability for the two processes 'Administering an exam' (BPMN model and checklist) and 'Apply for a business trip' (BPMN model, a short checklist and a long checklist).

the other evaluation criteria. There is hardly any spread in the responses concerning the checklists, which means that they provide a high reliability for nearly all test persons. Responses concerning the BPMN model again vary very much, but at a rather low level. Thus, even for the people who understood the BPMN evaluation tool quite well, it does not seem to provide an accordingly good value of reliability during the execution.

Concerning the expectations for the first case study, it can be said that practicability is attained simply by the fact that we were able to generate checklists out of BPMN models and that the processes were correctly performed. Also, comprehensibility was rated good or very good by the majority of the test persons and the correct performances of the processes support these statements.

7.3 Bank Case Study

The second case study was performed together with *Sparkasse Bamberg*, a German savings bank with 837 employees and a balance sheet total of EUR 3.752 billion (both in 2015) (Sparkasse Bamberg 2015). One process about how to handle an overdraft facility was chosen for the evaluation. This process is a very common one, i.e. we achieve a statistically sufficient number of process passes in a relatively short period of time. In a first step, the process model was transferred from an internally used modelling language to a BPMN model to provide comprehension of the process without having to introduce another modelling language to the reader and to make use of the transformation rules stated in Sect. 5 for generating the checklist. The BPMN model was validated by several employees including the CEO of the bank. For reasons of readability and secrecy, we shortened and translated the original model. This modified version of the overdraft facility process is shown in Fig. 28. The original model, a translated excerpt of it is shown in Fig. 29, consisted of 24 different activities, which were transformed into 14 tasks in the BPMN-model. Additionally, 6 exclusive gateways (splits and joins) and 4 events (start, end, abort) have been added. The length of the considered model, as defined in Sect. 7.2.1, is 24. The length of the shortened model in Fig. 28 is only 16.

Afterwards, the BPMN process model was transformed to a Process Checklist according to the rules from Sect. 5. This serialisation was done in merely about one hour. As the BPMN model was derived by hand from the internally-used process model, a BPMN-XML file was not given. Thus, the transformation to the checklist representation was done manually. The length of the resulting checklist is 19, consisting of 14 operating points and 5 control points. The checklist allows for 7 different cases to be executed, loops are not included. A summarising table of the objective criteria is given in Tab. 4.

The checklist bundle, i.e. the Process Checklist and the cover sheet, was then physically handed over to the employees of the savings bank to run the overdraft loan process. So far, only the internal process model was available as guideline for process execution. A manual of 2.5 pages was provided to learn about the usage of Process Checklists. The 31 employees who were involved in the overdraft loan process did not get any other instruction about how to use the Process Checklist. As in the case study in the academic domain, the employees filled in an evaluation sheet afterwards. Again, a Likert-scale with five classes was used to get their opinion about comprehensibility, orientation and reliability of the checklist. Furthermore, a fourth criterion we asked was their comprehensibility of the process so that participants not understanding the process itself could be excluded from the case study to prevent a bias of the results. Actually, this was not the case in our evaluation. The results of the evaluation are shown in Fig. 30 and summarised in the following.

At least 75% of the test persons claimed that they understand both the process and the checklist, i.e. its structure and its handling, quite well (the two higher classes of the Likert-scale). Note that the checklist was explained to the test persons only via a written instruction manual. Also, only a few hours after receiving the checklist and the manual, the employees went through the process for the

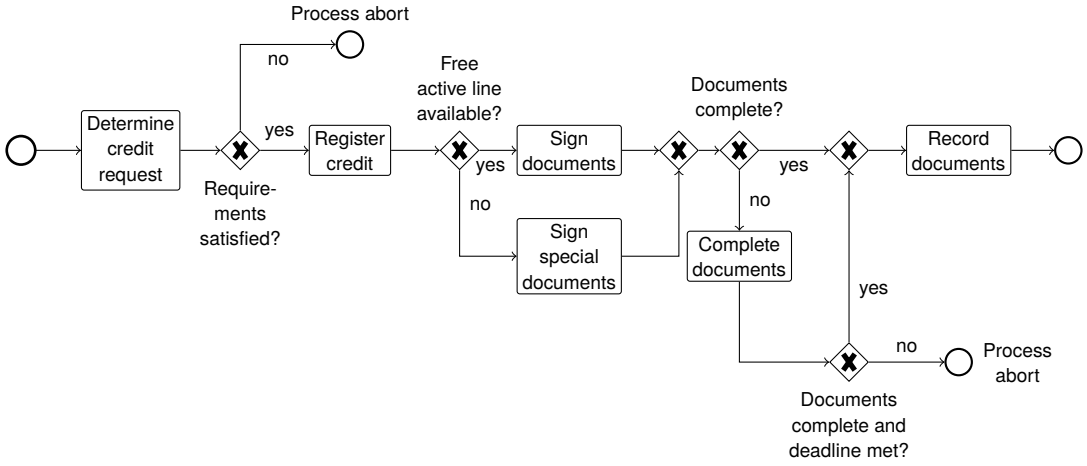

Figure 28: BPMN model of an overdraft facility process in a shortened version without data and human resources.

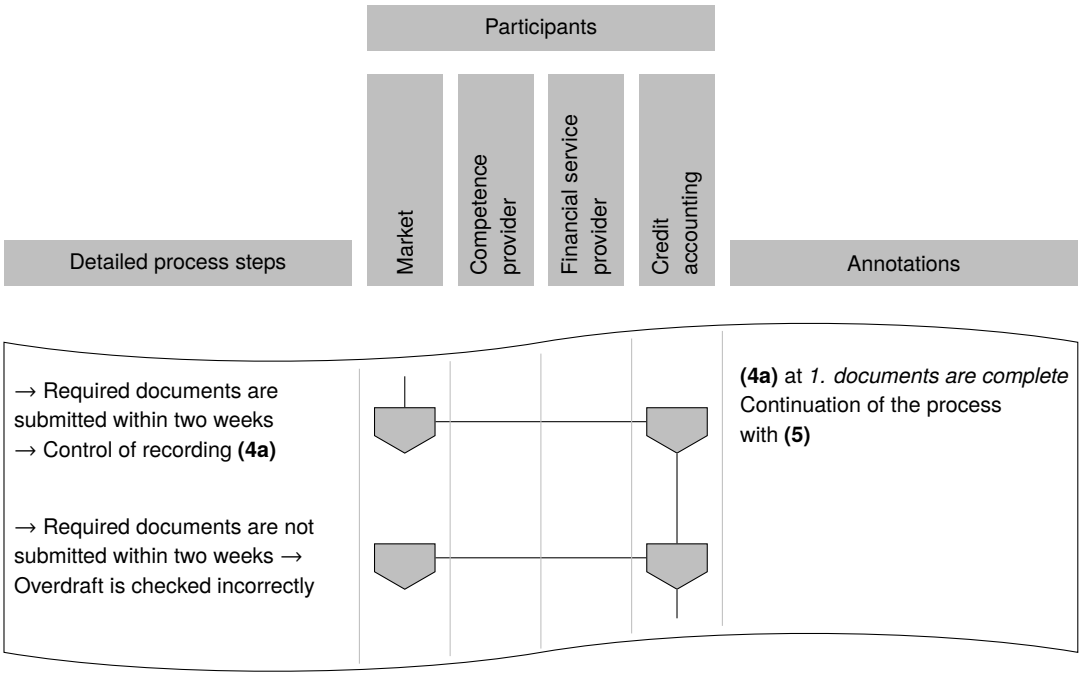

Figure 29: Translated excerpt of the overdraft facility process in its original notation.

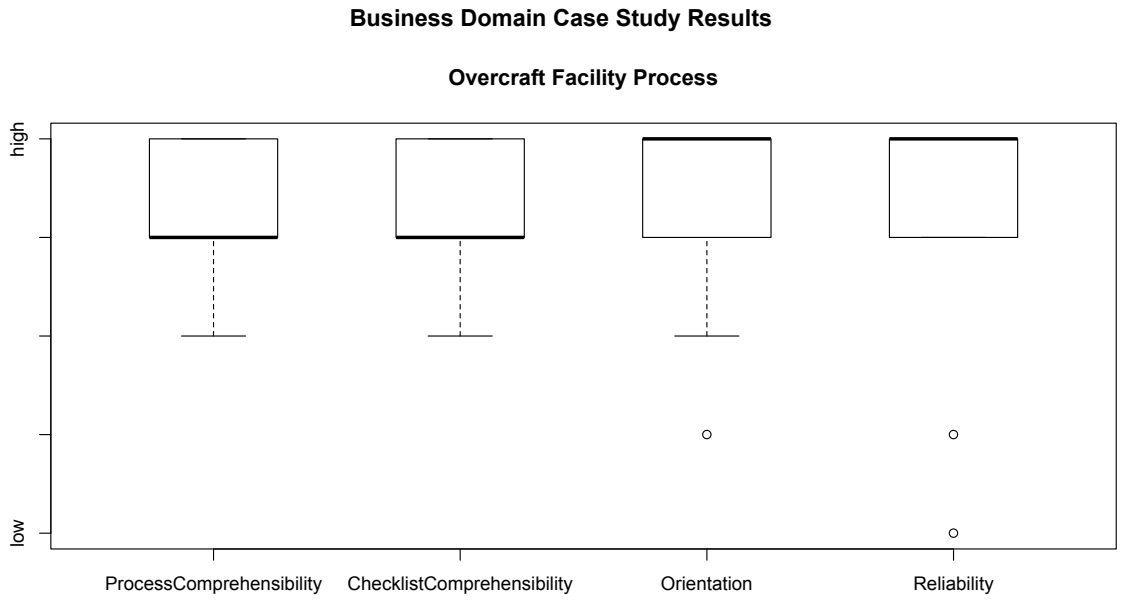

Figure 30: Results of the financial domain case study.

	Overdraft facility	
	BPMN	Checklist
Flexibility	low	high
Parallelism	*NA*	*NA*
Length	24	19

Table 4: Values for the objective evaluation criteria for the overdraft facility process support tools; NA = not available.

first time. No employee did not understand the process or the checklist (no responses in the two lowest classes). The results for the orientation criterion are slightly better, as at least 50% of the test persons even specified a very good (high) value for orientation and 75% a good or very good value. However, the assessment also shows some outliers for this criterion. The reliability criterion was, with a few exceptions, rated quite well. At least 50% assigned the highest value for reliability and at least 75% assigned at least the second highest value. During the evaluation, none of the employees deviated from the cases provided by the checklist.

These results of the subjective evaluation criteria basically seem to affirm our approach, but the most valuable outcomes are the (positive and negative) feedback comments and the proposals for improvement of the Process Checklist approach. Basically, the feedback and proposals could be divided into two categories: statements concerning the checklist and statements concerning the overdraft facility process. We show no further interest in the latter statements as they are process specific and do not relate to the checklist in the first place. The comments concerning the checklist can be divided into positive feedback, negative feedback and improvement proposals and are classified into several criteria. Issues declared as frequently are mentioned three or more times in the evaluation.

Positive Feedback

A frequent positive review is that *everybody understands the Process Checklist* or that *it is easy to understand the checklist*. Another positive review mentioned is that *traceability and transparency is rather high*, transparency in the sense of 'fully observable'. Furthermore, *no process step may be*

forgotten. Apparently, this last point also holds for any other WfMS. Also, traceability is an issue supported by WfMSs but Process Checklists provide easy access to this issue, particularly for the involved agents at every point in the checklist. The understanding of the checklist also strongly depends on the checklist design, especially on a clear arrangement of the checklist elements. Even if frequently mentioned as a positive feature, the checklist design could be improved to provide better comprehensibility to all employees.

Negative Feedback

Two frequently mentioned negative reviews are the following: *the processing of a Process Checklist is work and time intensive* and *the cover sheet is redundant*. For the first comment, some proposals for improvement are given in the following paragraph. Concerning the redundancy of the cover sheet a brief discussion is relevant. First of all, the cover sheet is redundant in such a way that a checklist point has to be signed in field *AG* and that its number has to be written on the cover sheet, so the completion of the task is indicated twice. The note on the cover sheet is, however, needed in that cases when the checklist points are not enacted sequentially, i.e. if jumps (forward or backward) are performed and so the point to be performed next is either not recognisable or only recognisable with a certain amount of reproducing efforts. In the following improvements section, a proposal for optimising the usage of the cover sheet is listed. Other characteristics negatively mentioned are that *the checklist is too complicated*, especially *the control points (gateways) are too confusing* and that *the checklist is too long*. We want to mention here that the length of the checklist strongly depends on the length of the underlying process. But also for this criticism a possible remedy is presented in the improvements section.

Improvement proposals

The most frequent proposal was to *use the Process Checklist only for long (complex) or infrequently executed processes*. *For simple and/or often executed processes, the checklist use offers no additional value but extends the processing time*. We would add another requirement for situations in which Process Checklists are not necessarily needed: The need for traceability is either not essential or can be achieved through other ways, like the archiving of documents produced through the process itself. Another improvement proposal relates to the extension of the processing time. The time requirement could be reduced by *not demanding complete signings with name, personnel number and so on, but to request only a short handwritten mark like initial letters (plus the personnel number)*. Adding the current date could be achieved by using a stamp. Concerning the usage of the cover sheet, it would be helpful to *list only the next point number on the cover sheet if the checklist is handed over to another agent*. If two or more points are consecutively executed by the same agent then the entry on the cover sheet is not needed. For reducing the length and therefore the time required for signing the points, it could be useful, if possible, to *summarise several small tasks that would have resulted in several checklist points into one checklist point if the assigned agent or role is the same and if the contents fit*. When doing so, the change of checklist elements could become a little costlier because the composed elements have to be separated first before applying the changes (and summarised afterwards again). For generating the Process Checklist for the financial domain evaluation, we simply used the same task allocation as given through the underlying (internal) graphical process model, which showed a relatively fine granularity. Concerning the design of the Process Checklist it was proposed to *highlight the middle field of each checklist point* (the activity field *AC* of operating points and the condition field *CO* of control points) and to *clearly identify operating and control points through signal words like 'task' and 'decision'*. Also, another remark given by the test persons was to *process the checklist electronically*. This proposal is justified in times of mobile devices but we wanted to examine and test paper-based Process Checklists

explicitly for the reasons given in Sect. 1: their fast, cheap and easy enactment. However, the structure of Process Checklists as introduced in Sect. 2 may likely be transferred to an electronic version of the Process Checklist. This would be an issue for future work.

To sum up, we can say that comprehensibility was rated well by over 3/4 of the test persons, which is a very high percentage after one test run. Furthermore, the comments show a basic acceptance and a certain necessity for checklists by most of the employees. However, the improvement proposals should be considered in any case and incorporated as far as possible to further increase the acceptance. As the improvement proposals show, the necessity for checklists strongly depends on the types of processes. Complex or infrequently performed processes require execution support.

At the end of this paper, a short conclusion will follow, highlighting also the restrictions of the proposed Process Checklist approach, as well as issues for future work that could be devoted to this topic.

8 Conclusion, Limitations and Future Work

In order to provide an alternative to IT-based process management systems, the work at hand presented a paper-based scheme in order to support workflow execution that is suitable for human-driven processes. We introduced the Process Checklist representation of process models where processes are described as a paper-based step-by-step instruction manual. The Process Checklist is handed over during process execution from process participant to process participant. Successful task accomplishments are recorded with the help of signatures of corresponding process participants.

In this way, the Process Checklist also supports the key benefits of traditional WfMSs. The checklist is handed over to responsible agents (task coordination), process tasks are serialised and marked by a unique identifier (step-by-step guidance) and the checklist itself as well as the corresponding signatures ensure traceable process execution. The work at hand provides the following as the main contributions:

- the general structure of Process Checklists to serve as an execution support
- enactment instructions for Process Checklists
- a transformation algorithm of basic BPMN process model elements to Process Checklists
- a comprehensive evaluation

Furthermore, we described implementation details by giving a concrete checklist meta-model as well as an XML-based serialisation possibility. The checklist approach has been evaluated in two real-life case studies, one in the academic domain and one in the financial business domain. The results show that Process Checklists serve as a feasible process execution support and are highly accepted by process participants.

Paper-based checklists also have disadvantages compared to traditional WfMS. Checklists represent a single point of access, so support for distributed agents may be difficult. If this is the case, one has to ask if using a paper-based checklist is the right thing for this specific application, as we recommend using checklists, for example, in processes where information is mostly present on paper or the physical realm and moving this information into the digital realm is prohibitively costly or even impossible. Moreover, one disadvantage may occur due to human failure as the Process Checklist, i.e. one process instance, simply may get lost. Ameliorating such a circumstance can be effected with the help of receipts, though this task may be laborious. However, the problem of losing documents is not only an issue concerning paper-based checklists, but is relevant for all institutions dealing with documents and files, e.g. in accounting.

In general, it is possible to transform a procedural process model to a Process Checklist based on the proposed algorithm. However, due to the serialisation of the process, the checklist representation has of course problems when dealing with parallelism. Here, process modellers have

to choose a suitable transformation method as described in Sect. 5. The presented case studies focused on a first evaluation in the fields of university processes and banking processes. Here, we got useful information regarding the acceptance and cooperation of participating agents as well as valuable suggestions for improving methodology, design and representation.

A further extension of the checklist approach, as it was also proposed by test persons in the case studies, would be to investigate if it is possible to use the transformation rules from BPMN process models to checklists as the basis for a digital ToDo application, e.g. for mobile devices to achieve better navigation through the process instance than the paper-based checklist provides. In doing so, the problem of documents and files that need to be passed—now detached from the checklist—will have to be addressed again. Furthermore, we will focus the transformation of loosely-specified process models like declarative process models defined in languages like Declare (Baumann et al. 2016; Pešić 2008) or DPIL (Zeising et al. 2014), which already contain a high degree of flexible process execution but need, due to this extreme flexibility, appropriate navigation and support tools.

References

AXONiVY (2016) Axon.ivy BPM Suite - Axon.ivy Designer. http://developer.axonivy.com/download/. Last Access: 2017-01-11

Baumann M., Baumann M. H., Gruber D. F.-X., Jablonski S. (2016) Infinite Horizon Decision Support For Rule-based Process Models. In: International Journal on Advances in Software 9(1&2), pp. 141–153

Baumann M., Baumann M. H., Schönig S., Jablonski S. (2014) Enhancing Feasibility of Human-Driven Processes by Transforming Process Models to Process Checklists. In: Bider I., Gaaloul K., Krogstie J., Nurcan S., Proper H. A., Schmidt R., Soffer P. (eds.) Enterprise, Business-Process and Information Systems Modeling. LNBIP Vol. 175. Springer, Berlin, Heidelberg, pp. 124–138

Boehm B. (1991) Software risk management: principles and practices. In: Software, IEEE 8(1), pp. 32–41

Briere J., Johnson K., Bissada A., Damon L., Crouch J., Gil E., Hanson R., Ernst V. (2001) The Trauma Symptom Checklist for Young Children (TSCYC): reliability and association with abuse exposure in a multi-site study. In: Child Abuse & Neglect 25, pp. 1001–1014

C2P2 (2015) Competence Center for Practical Process Management. http://www.kppq.de/. Last Access: 2017-01-11

Chong S. (2014) Business process management for SMEs: an exploratory study of implementation factors for the Australian wine industry. In: Journal of Information Systems and Small Business 1(1-2), pp. 41–58

Condon C. (1993) The Computer Won't Let Me: Cooperation, Conflict and the Ownership of Information. In: Easterbrook S. (ed.) CSCW: Cooperation or Conflict? CSCW. Springer, London, pp. 171–185

Degani A., Wiener E. L. (1991) Human factors of flight-deck checklists: the normal checklist. In: NASA Contractor Report 177549

Derogatis L. R., Lipman R. S., Rickels K., Uhlenhuth E. H., Covi L. (1974) The Hopkins Symptom Checklist (HSCL): A self-report symptom inventory. In: Behavioral Science 19(1), pp. 1–15

Ely J. W., Graber M. L., Croskerry P. (2011) Checklists to reduce diagnostic errors. In: Academic Medicine 86(3), pp. 307–313

Faerber M., Jablonski S., Schneider T. (2007) A Comprehensive Modeling Language for Clinical Processes.. In: ECEH. Citeseer, pp. 77–88

Fahrmeir L., Heumann C., Künstler R., Pigeot I., Tutz G. (2016) Statistik – Der Weg zur Datenanalyse, 8th ed. Springer, Berlin, Heidelberg

GPM Netzwerk (2015) Netzwerk Geschäftsprozessmanagement. http://www.gpm-netzwerk.de/prozessmanagement-in-kmu.html. Last Access: 2017-01-11

Hales B. M., Pronovost P. J. (2006) The checklist – a tool for error management and performance improvement. In: Journal of Critical Care 21(3), pp. 231–235

Harrison-Broninski K. (2010) Dealing with Human-Driven Processes. In: vom Brocke J., Rosemann M. (eds.) Handbook on Business Process Management 2: Strategic Alignment, Governance, People and Culture. Springer, Berlin, Heidelberg, pp. 443–461

Hartel M. C., Chou S. C. (1995) Electronic Checklist System. United States Patent (Patent Number 5 454 074)

Hauser R., Friess M., Kuster J., Vanhatalo J. (2006) Combining Analysis of Unstructured Workflows with Transformation to Structured Workflows. In: Enterprise Distributed Object Computing Conference, pp. 129–140

Hevner A., Chatterjee S. (2010) Design Science Research in Information Systems. In: Hevner A., Chatterjee S. (eds.) Design Research in Information Systems: Theory and Practice. Springer, New York, London, pp. 9–22

Jablonski S. (2010) Do We Really Know How to Support Processes? Considerations and Reconstruction. In: Engels G., Lewerentz C., Schäfer W., Schürr A., Westfechtel B. (eds.) Graph Transformations and Model-Driven Engineering. LNCS Vol. 5765. Springer, Berlin, Heidelberg, pp. 393–410

Jablonski S., Bussler C. (1996) Workflow Management: Modeling Concepts, Architecture and Implementation. International Thomson Computer Press, London

Jackson M. A. (1975) Principles of program design. Academic Press, Orlando, FL

Kellerhoff F. (2012) Prozessanalyse und Prozesskostenrechnung im Krankenhaus - Ergebnisse aus der Praxis. In: Mühlbauer B. H., Kellerhoff F., Matusiewicz D. (eds.) Zukunftsperspektiven der Gesundheitswirtschaft. Lit, Berlin, Münster, pp. 64–82

Kerzner H. R. (2013) Project Management: A Systems Approach to Planning, Scheduling, and Controlling, 11th ed. John Wiley & Sons, Hoboken, NJ

Koehler J., Hauser R., Küster J., Ryndina K., Vanhatalo J., Wahler M. (2008) The Role of Visual Modeling and Model Transformations in Business-driven Development. In: Electronic Notes in Theoretical Computer Science 211 Proceedings of the Fifth International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT 2006), pp. 5–15

Kumar A., Zhao J. L. (2002) Workflow support for electronic commerce applications. In: Decision Support Systems 32(3), pp. 265–278

Luff P., Heath C., Greatbatch D. (1992) Tasks-in-interaction: Paper and Screen Based Documentation in Collaborative Activity. In: Computer-supported Cooperative Work. CSCW. ACM, Toronto, Ontario, Canada, pp. 163–170

Mans R., van der Aalst W. M. P., Russell N., Bakker P. (2009) Flexibility Schemes for Workflow Management Systems. In: Ardagna D., Mecella M., Yang J. (eds.) Business Process Management Workshops. LNBIP Vol. 17. Springer, Berlin, Heidelberg, pp. 361–372

Melenovsky M. J. (2005) Business process management's success hinges on business-led initiatives. In: Gartner Research, Stamford, CT July, pp. 1–6

Montali M. (2009) Specification and Verification of Declarative Open Interaction Models – A Logic-based framework. PhD thesis, Alma Mater Studiorum Universitá di Bologna

Nassi I., Shneiderman B. (1973) Flowchart techniques for structured programming. In: ACM Sigplan Notices 8(8), pp. 12–26

Object Management Group Inc. (2011) Business Process Model and Notation (BPMN) Version 2.0. http://www.omg.org/spec/BPMN/2.0. Last Access: 2017-01-11

Organisationshandbuch (2015) Das Organisationshandbuch - vom Praktiker für den Praktiker. http://www.orghandbuch.de. Last Access: 2017-01-11

Pešić M. M. (2008) Constraint-based Workflow Management: Shifting Control to Users. PhD thesis, Technische Universiteit Eindhoven

Reichert M., Weber B. (2012) Enabling flexibility in process-aware information systems: challenges, methods, technologies. Springer, Berlin, Heidelberg

Schonenberg H., Mans R., Russell N., Mulyar N., van der Aalst W. M. P. (2008) Process Flexibility: A Survey of Contemporary Approaches. In: Dietz J., Albani A., Barjis J. (eds.) Advances in Enterprise Engineering I. LNBIP Vol. 10. Springer, Berlin, Heidelberg, pp. 16–30

Seitz M., Schönig S., Jablonski S. (2014) A Framework for Reasonable Support of Process Compliance Management. In: Abramowicz W., Kokkinaki A. (eds.) Business Information Systems Workshops. LNBIP Vol. 183. Springer, New York, pp. 131–144

Sparkasse Bamberg (2015) Report 2015. https://www.sparkasse-bamberg.de/de/home/ihre-sparkasse/ihre-sparkasse-vor-ort/finanzpublikationen.html?n=true. Last Access: 2017-01-11

Swenson K. D. (2010) Mastering the Unpredictable: How Adaptive Case Management Will Revolutionize the Way That Knowledge Workers Get Things Done. Meghan-Kiffer Press, Tampa

The R Foundation (2017) R: The R Project for Statistical Computing. http://www.r-project.org/. Last Access: 2017-01-11

van der Aalst W. M. P., ter Hofstede A. H. M., Kiepuszewski B., Barros A. P. (2003) Workflow Patterns. In: Distributed and Parallel Databases 14(1), pp. 5–51

van der Aalst W. M. P., Weske M., Grünbauer D. (2005) Case handling: a new paradigm for business process support. In: Data & Knowledge Engineering 53(2), pp. 129–162

WHO (2009) WHO Guidelines for Safe Surgery 2009: Safe Surgery Saves Lives. http://www.ncbi.nlm.nih.gov/books/NBK143244/. Last Access: 2017-01-11

Wolff A. M., Taylor S. A., McCabe J. F. (2004) Using checklists and reminders in clinical pathways to improve hospital inpatient care. In: Medical Journal of Australia 181, pp. 428–431

Zairi M. (1997) Business process management: a boundaryless approach to modern competitiveness. In: Business Process Management Journal 3(1), pp. 64–80

Zeising M., Schönig S., Jablonski S. (2014) Towards a common platform for the support of routine and agile business processes. In: International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), pp. 94–103

zur Muehlen M., Recker J. (2008) How Much Language Is Enough? Theoretical and Practical Use of the Business Process Modeling Notation. In: Bellahsène Z., Léonard M. (eds.) Advanced Information Systems Engineering. LNCS Vol. 5074. Springer, Berlin, Heidelberg, pp. 465–479