Preprint Version 2

The Process Checklist – Simple Enactment of Human-driven Processes

Transforming Business Processes to Process Checklists

Michaela Baumann · Stefan Schönig · Michael Heinrich Baumann · Stefan Jablonski

2015

Abstract In traditional approaches business processes are executed on top of IT-based Workflow-Management Systems (WfMSs). However, when dealing with human-driven workflows, conventional WfMSs may turn out to be too restrictive. Especially, the only way to handle exceptions is to bypass the system. If users are forced to bypass WfMSs frequently, the system is more an obstacle than an asset. In order to diminish the dependency from IT-based process management systems, we propose a cheap and fast installable alternative way of supporting workflow execution that is especially suitable for human-driven processes. We introduce the so-called Process Checklist representation where

M. Baumann
Databases and Information Systems
University of Bayreuth
Tel.: +49-921-55-7625
Fax: +49-921-55-7622
E-mail: michaela.baumann@uni-bayreuth.de

S. Schönig
Databases and Information Systems
University of Bayreuth
Tel.: +49-921-55-7627
Fax: +49-921-55-7622
E-mail: stefan.schoenig@uni-bayreuth.de

M. H. Baumann
Applied Mathematics
University of Bayreuth
Tel.: +49-921-55-3280
Fax: +49-921-55-5361
E-mail: michael.baumann@uni-bayreuth.de

S. Jablonski
Databases and Information Systems
University of Bayreuth
Tel.: +49-921-55-7620
Fax: +49-921-55-7622
E-mail: stefan.jablonski@uni-bayreuth.de

processes are described as a paper-based step-by-step instruction handbook. Process Checklists can either be set up directly for the desired process or be derived from an already existing process model. A transformation from BPMN models to Process Checklists is given as well as implementation instructions with an underlying meta-model. Furthermore, evaluations in the academic as well as in the business domain have been performed to evaluate the approach.

Keywords process modelling · Process Checklists · paper-based process execution

1 Introduction

For approximately 20 years process management is regarded as an innovative technology both for the description of complex applications and for supporting their execution [14]. In traditional approaches business processes are executed on top of IT-based Workflow-Management Systems (WfMSs) [31]. The key benefits of the application of a WfMS are task coordination, step-by-step guidance through process execution and traceability supporting compliance issues [25].

However, when dealing with human-driven workflows that heavily depend on dynamic human decisions, conventional WfMSs turn out to be too restrictive [1]. Especially, the only way to handle exceptions – which regularly occur in human-driven workflows – is to bypass the system. If users are forced to bypass WfMSs frequently, the system is more an obstacle than an asset [1]. In total, users start to complain that "the computer won't let them" to do the things they like to accomplish [5]. So users like to get more independent from "electronic systems" in order to become more flexible. If original documents are needed for executing a process, in many cases a paper-based execution model is preferred [19]. Furthermore, the introduction of a WfMS is regarded as a huge and cost-intensive project [21] that many small and medium-sized companies dread to establish as they cannot afford to introduce such a system. However, they desire to manage their processes since they regard them as valuable and effective.

In order to diminish the dependency from IT-based process management systems, we propose an alternative way of supporting workflow execution that is especially suitable for human-driven processes, like it is the case for example in public administration, authorities and service companies. We introduce the so-called Process Checklist representation of process models which is a fast way of setting up process execution support almost on the fly. Installation time is virtually nonexistent and needs for structural changes both in the introduction phase and in the operating phase can be applied immediately and rapidly. Furthermore, the Process Checklist is independent from specific hardware and operating systems.

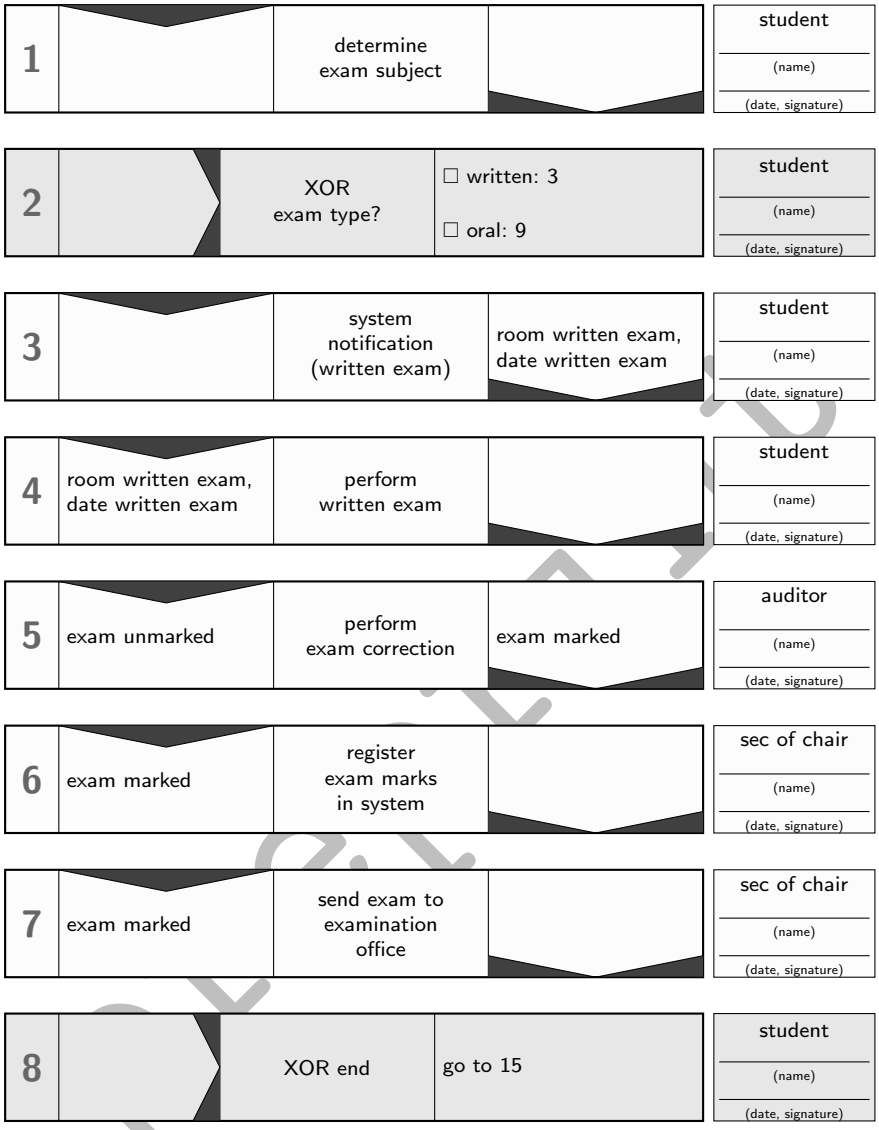

Fig. 1: First part of the graphical checklist for the process "subscribing for an exam".

In the presented Process Checklist approach, processes are described as a paper-based step-by-step instruction handbook. The Process Checklist is handed over during process execution from process participant to process participant. Successful task accomplishments are recorded through signatures of corresponding human agents. In principle the most important outcome is that at the end of the process all required signatures are on the checklist. So it is completely output oriented. Nevertheless, the checklist method describes a valuable form of process usage and widens its spectrum towards non-computer based and flexible and cheap process execution. If needed, certain parts of the checklist can even be deleted, changed or added during the execution by simply using a pen. However, traceability is still maintained. Nevertheless, there is a prerequisite to achieve traceability: 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. In our three sample scenarios presented in Section 7 it was no issue to enforce this regulation.

Besides, a Process Checklist also supports the key benefits of traditional WfMSs. The checklist is handed over to responsible agents (task coordination), process tasks are serialized 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 general structure of Process Checklists as well as an elaborate transformation algorithm of basic Business Process Model and Notation (BPMN) process model elements [23] to Process Checklists. Note, that this article is an extended paper of a conference proceeding [2]. In addition to various improvements and further concepts the work at hand extends our previous article by a description of the actual implementation (Section 6) as well as by a detailed evaluation and case study section (Section 7).

2 Structure of the Process Checklist

According to [15], there are five perspectives a process model should cover and that a Process Checklist should therefore support, too, to serve as a sophisticated support tool for process execution: There are the functional perspective (i.e., the task description), the organizational perspective (i.e., the assignment of agents to tasks), the data perspective that describes data flow, including data generation and consumption, the operational perspective (i.e., the assignment of systems and services to tasks) and the control flow aspect. 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 there are two kinds of checklist steps, also called checklist points in the further course: operating points that describe the activities of a process and control points that 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 coloured light and control points are coloured dark (in Fig. 1, step no. 2 and 8). The two different kinds of checklist points are explained in the following.

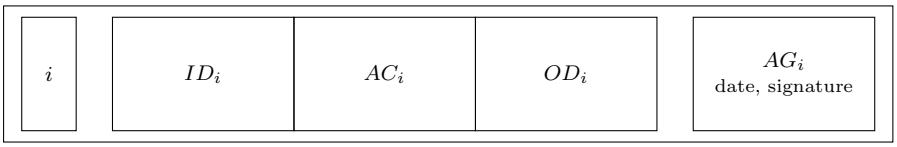

Fig. 2: Schematical representation of an operating point.

A schematical representation of an operating point is given in Fig. 2. An operating point has five fields that need to be filled in with information about a certain activity. Field i on the left-hand side assigns a unique number 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 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. Apparently, an operating point contains information about the functional (field AC_i), organizational (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.

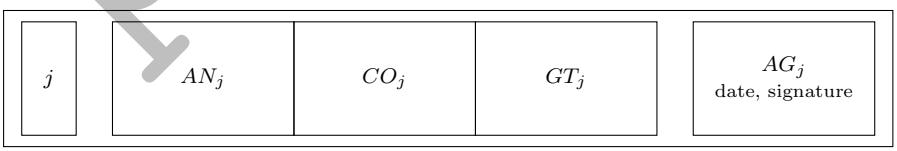

Fig. 3: Schematical representation of a control point.

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.

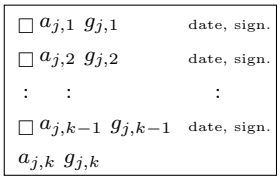

Fig. 4: Schematical representation of field GT_j of control point No. j.

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. Variables $a_{j,\cdot}$ 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 indepedently then variables $a_{j,\cdot}$ are simply replaced by the phrase "go to". Also, the names of subchecklists may be listed there. Variables $g_{j,\cdot}$ refer to the numbers of other checklist points, whereto the checklist has to be passed in the correpsonding 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 variable c which is $c_2 = 0$ in the case of Fig. 1. Variable c also determines whether the last (a, g)-pair (in Fig. 1 pair $(a_{j,k}, g_{j,k})$) where a is usually replaced by "finally go to" and the box □ is not visible, is present. It is present in case $c = 1$. Why this confirmation line, the last (a, g)-pair and all other components are needed will be further explained in Section 4.

To conclude this section, a fully formalized 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.

Definition 1 (Checklist Vector) A checklist is a vector $\mathcal{C} = (p_1^t, p_2^t, \ldots, p_n^t)$, $n \in \mathbb{N}$, $t \in \{o, c\}$ 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

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

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

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		student
5	o	{exam unmarked}	perform exam correction	{exam marked}	auditor
6	o	{exam marked}	register exam marks in system		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

Fig. 5: Checklist vector for the process "Subscribing for an exam" containing all elements according to Definition 1 needed for the graphical checklist.

The checklist vector representation of the sample process "subscribing for 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 but 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 [30]. 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 [11], aircraft preparation [6] [12], or project management [3], to mention only a few examples. Usually, in all of these fiels checklists are rather an unsorted list of application specific items that have to be checked for validity. 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., [7] [4]). Roughly summarized, symptoms are manually recorded according to a list of possible symptoms and then, analyzing 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 analyzing the patient's health condition itself. Research in this application field of checklists is mainly done in the medical area like answering the question of what are the right questions to ask to detect specific illnesses or to exclude them. Furthermore, in the clinical area, [9] proposes the use of checklists to compactly map small process steps that can be executed in arbitrary order within a process model. That means, 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 organizational issues.

3.2 Aircraft Crew Checklists

Checklists in the field of air traffic mainly serve as a remainder to obtain a proper configuration of the plane and full quality and security in every flight [6] [12]. 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 and 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 [6] mentions, there are also several disadvantages of paper checklists. This holds even if they would have to be used as enactment means for processes. 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, like scroll checklists, mechanical or electromechanical checklists or even vocal checklists. Of course, computer-aided checklists are also used mainly for normal cockpit tasks like takeoff and landing where the pilot needs both hands to fulfill the single process steps. However, all types of checklists that are mentioned in this context lack some of the process perspectives listed at the beginning of Section 2. Apart from the functional perspective, no other perspectives are fully visible on aircraft paper checklists. The control-flow perspective is induced for some tasks that have to be executed in a determined order by the ordering of the tasks on the checklist stringently top-to-bottom. Systems and services are usually prescribed in an airplane, e.g., for checking hydraulic pressure the corresponding measuring instrument and display have to be used. Responsible agents are usually appointed vocally by the pilot during the execution. The data perspective is not mentioned. In [6] also some guidelines for designing and using flight-deck checklists are listed, like the ordering of checklist items either according to certain dependencies or arbitrarily and the consideration of the workload of human agents when assigning the different tasks. 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 that great importance to us as in [6].

3.3 Project Management Checklists

Checklists in the field of project management can either help organizing the project team members or serve as identifiers, e.g., risk identifiers [3]. 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 [16]. Checklists helping organizing the team are task lists that allow to assign team members to the task and to distiguish between mandatory and optional items [16]. These task lists are however customized to each project, rarely cover whole processes and do not follow any standardization. They also do not contain information about the five process perspectives expect for the functional one. But they serve as a good starting point, together with the other presented types of checklists, for the Process Checklist repre-

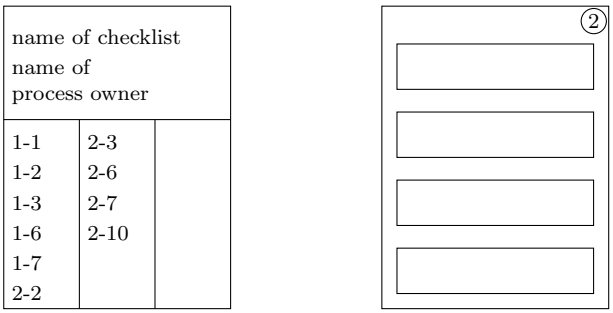

Fig. 6: Cover sheet (left-hand side) with name of the checklist/process, name of process owner and a list of already executed steps and one point to be executed next; checklist (right-hand side) with current number in the upper right corner and several operating/control points. Note, that in this fictional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

sentation formulated in the work at hand.

In this paper, we try to bring the widely used unstructured checklists that usual do not provide any information about execution order and other execution modalities like parallelism or exclusiveness into a more structured format to receive a structured process execution tool but also to maintain the uncomplicated usage of paper-based checklists. In [14] and [27] this kind of checklist is suggested but not elaborated. The disadvantages listed in Section 3.2 are either not essential when it comes to business Process Checklists (like the occupancy of one hand) 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 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 Section 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 so that the elements can be arranged in a suitable way.

4 Enactment of the Graphical Checklist

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. Depeding 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. This list of points serving as state indicator induces a pointer which is listed as one of the missing checklist elements in [6]. The cover sheet and the graphical checklist together form the basis of the executable Process Checklist bundle. In fact, there may be also more than one checklists attached to the checklist bundle as it is possible that some points need to be proceeded more than once. If this is the case, i.e., a so-called backward jump is necessary, 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 the figure, 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), also 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 course so far 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.

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, that means input documents, have to be added and scheduled in the respective list. On the cover sheet “1–1” is noted, that means, 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, that means especially if all listed documents are handed over, and quit the delivery.

The process owner has to archive the signed receipt for later reconstruction if necessary.

When an agent gets the graphical checklist he has to run through this acknowledgement process (checking 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 (like in Fig. 6) he has to look at point 10 of the current checklist, that 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. After execution of the current point the agent has look which agent is next. If it is himself he executes the next point and writes it down on the cover sheet, else 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.

4.1 Execution of Operating Points

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

4.2 Execution of Control Points

Control points can have different characteristics as they represent, for example, exclusive, inclusive or parallel splits. These three terms that 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 advises that are necessary when certain points have to be skipped or for joining the distribution of multiple subchecklists.

4.2.1 Execution of Exclusive Splits

If a control point marked, e.g., 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 □ in front of the corresponding answer, e.g., the

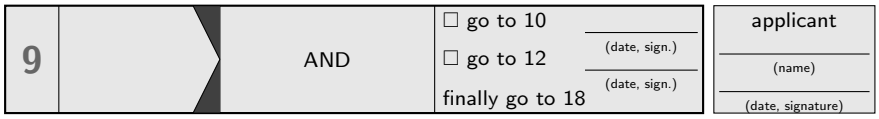

Fig. 7: A control point suitable for dynamic sequential execution

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: $g_{\cdot,l}$ is greater than the current point number, then everything can go on as before. 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 recognize 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. When there are no backward jumps over the whole checklist the consecutive checklist number is not needed.

4.2.2 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 several possibilities to do so. One method where only one checklist is needed is the dynamic sequential execution which allows for choosing a suitable order of the subprocesses. Parallelism is dropped, however. The postbox method retains parallelism but relaxes the typical checklist properties. Parallel execution also keeps parallelism but requires separate checklists for all subprocesses. The three methods are explained more detailed in the following.

Dynamic Sequential Execution When coming to a 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 points is shown in Fig. 7. Note the additional go-to-number in field GT that refers to the checklist point that has to be executed after all the subprocesses have been executed successfully. When he 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 of the branch was made. So, the order deciding agent 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 conformed 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 only consists of one point, point No. 10, as point No. 11 needs to be a control point referring back to the order decision point No. 9. The second subprocess starts at point No. 12 and ends at point No. 16. Point No. 17 is again a control point referring back to point No. 9. After both subprocesses have been successfully executed the branches are joined in point No. 18 that 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 (Section 4.2.1). 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, variable *c* of the checklist vector definition is set to 1, i.e., the additional confirmation lines and a finally-go-to-number without marking box ☐ 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. 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. With this 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 documents now being in the postbox, checks for completeness, signs in *AG* if everything is okay and goes on as before. This method may become confusing if too many agents are involved in the subprocesses and needs coordination, initiative and especially individual responsibility of all agents. But it provides a possibility of considering the parallelism aspect of the several subbranches.

Parallel Execution With this method named parallel execution it is also possible to consider simultaneity of the different branches like it is the case for the postbox method but the execution itself takes place in a more structured and checklist-like way. The parallel execution of parallel splits also 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. A finally-go-to-number passes on to the next point of the main checklist.[1] 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 somehow 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, visualized with BPMN modelling tools. Here, the two long subprocesses $l1$ and $l2$ can be executed with two separate checklists whereas the short subprocess can be included into the checklist in a (dynamic) sequential way, i.e., it would be performed before or after the two long subprocesses.

4.2.3 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. Like for exclusive splits, certain conditions may be fulfilled, but 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 inlcuded into the checklist. Possible anwers 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 into 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

[1] Through setting variable c to $c = 1$ in the control point distributing the subchecklists, the finally-go-to instruction is visible in field *GT*. However, this instruction is not needed as the next point in the main checklist is executed either way if no separate jump instruction was given.

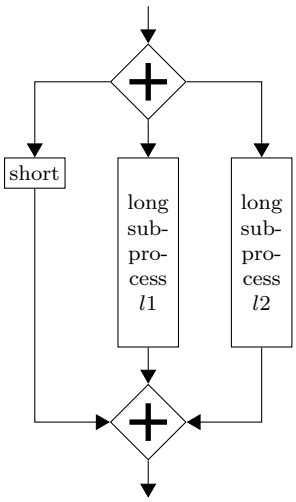

Fig. 8: Schematic part of a process model with one short and two comparatively long parallel subprocesses (i.e., consisting of a lot of tasks) that can be executed as a composition of presented methods.

be included into the ckecklist 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 it is the case for the sequential dynamic method, are needed. All subprocesses that shall be executed have to be marked in the box $\square$ in field GT, at least one box up to all boxes. For the marked boxes and the corresponding subprocesses it is proceeded 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 which activities need to be executed and which not.

Note, that for parallel and inclusive operating points no backward jumps are possible. Also, 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 lacks 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 has to weigh which execution method is best for each parallel or inclusive split situation.

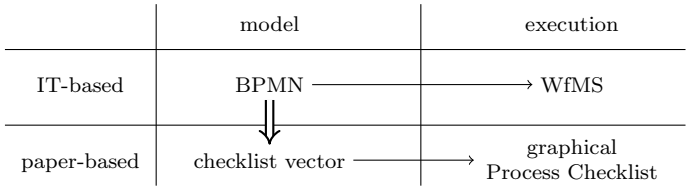

Fig. 9: Schematic approach to differentiate paper-based from IT-based process management systems.

5 Transformation of Process Model Elements through Serialization

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. In many organizations process models for nearly all organizational activities exist but they are often not supported by execution tools. In this section we want to show that an execution support tool, namely the paper-based Process Checklist, can fast and easily be generated out of existing process models, where BPMN should just be seen as an illustrative example for process model languages the existing process models may be available in. In the following it is explained, in which way the single elements of the (BPMN) process model are transferred into either operating points or control points, i.e., are serialized. 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., [13] and [17]. 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 the previously mentioned paper [2] which is the basis for the work at hand.

The conversion steps are virtually performed in a simply algorithmic way, except for parallel and inclusive gateways. There, the modeller has to decide which of the execution methods presented in Section 4.2.2 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 like. These specifications are necessary to give concrete mapping rules. For process models, only basic elements of the Business Process Modeling Notation are allowed, as [22] shows this is enough in most cases and as the paper at hand has to be seen as first expanded approach to this topic.

Definition 2 (BPMN Process Model) A process model is defined according to the Business Process Model and Notation (BPMN) 2.0 (see e.g. in [23]) allowing for the following basic elements:

– flow objects: activities, events (start, end), gateways (AND, XOR, OR)

– sequence flows
– data (input/output) objects
– participants: one pool, possibly separated into different lanes

Further on, it needs to be block-structured (see, e.g., [18]) to guarantee soundness of the model and a correct checklist representation.

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 like done in one case study shown in Section 7.2 and Fig. 27, we simply have not specified a standardized 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". Besides, message flows are not part of all process modelling languages and also not one of the basic elements of BPMN [22].

Non-block-structured process models can be transformed to Process Checklists, too, but the specification of a universal algorithm for this is difficult and case-dependent to get a valid and sound checklist representation which is why we state this restrictive requirement in Definition 2. 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 according to Definition 2 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 description are mapped on the field AC whereas all directly incoming data and directly outgoing data is mapped on the field ID and OD respectively. The participant of the corresponding lane or hierarchy of lanes, that may be a single person or a role specification is mapped onto the field AG. Concrete agent names are filled in during the execution of the checklist. An example of an activity with documents and participant is given in Fig. 10 and the corresponding checklist point, an operating point, in Fig. 11.

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:

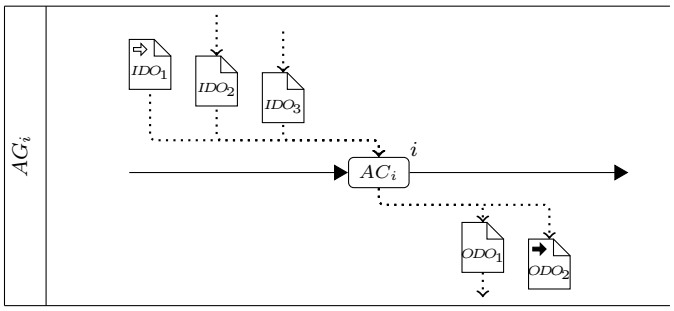

Fig. 10: Abstract BPMN representation of an activity with ingoing and outcoming documents.

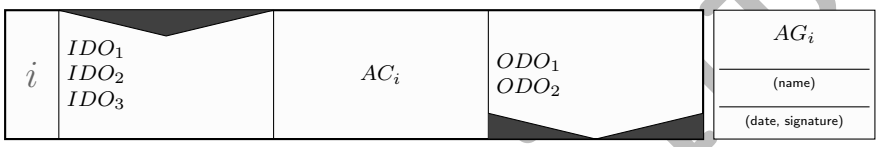

Fig. 11: Representation of the activity of Fig. 10 as an operating point of a checklist.

1. Include the complete subprocess. This leads to a comparatively long but correct 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 Y, go to:", $g_{j,1} = 1$) which has to be confirmed due to $c_j = 1$ and with instructions for waiting for this checklist to come back completely processed. Agent Y is one agent of the role description of the first point in the subchecklist. Parameter $a_{j,2}$ is set to "finally go to" and $g_{j,2} = j + 1$.[2] Agent y is the agent of the first task of the subprocess. This is according to the multiple checklist method for parallel splits presented in Section 4.2.2, 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 done 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.

5.3 Transformation of Gateways

As already the designing and execution of splits and joins of a directly generated checklist imposes several possiblities, also the transformation of BPMN

[2] Not needed here when simply regarding the sequence flow, but due to $c_j = 1$ the final-go-to-instruction is visible.

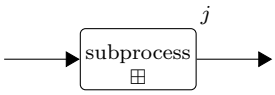

Fig. 12: Symbol for a subprocess in BPMN 2.0.

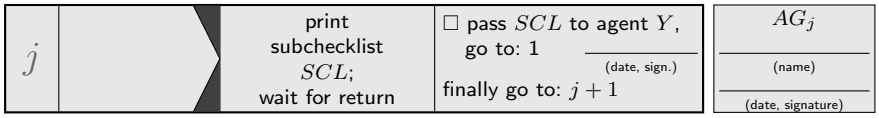

Fig. 13: Representation of the subprocess task of Fig. 12 according to the second method (distributing a subchecklist) as a control point of a checklist.

gateways to checklist elements does so. Most of the methods described in the following correspond to one execution method presented in Section 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 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. The control point is proceeded 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 (the join exists because of the demanded block-structure), 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 Section 4.2.1. Only the branch listed last in the checklist does not need this jump instruction as the point after the exclusive join is performed next, anyway. 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). Because of the block-structure, a backward-jump split gateway has only two outgoing edges, one pointing forward and one backward. In this case, the join gateway before the split gateway does not need to be considered. 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.

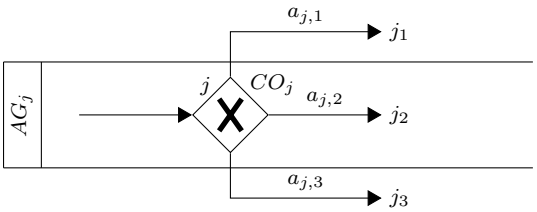

Fig. 14: Exclusive split gateway without a backward jump.

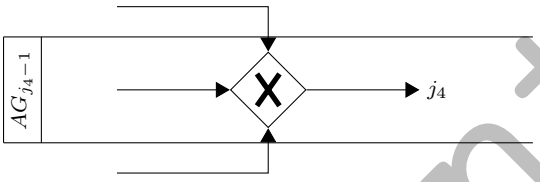

Fig. 15: Exclusive join gateway (without backward jump) that does not have to exist if the outgoing branches of the exclusive split gateway end with terminal events.

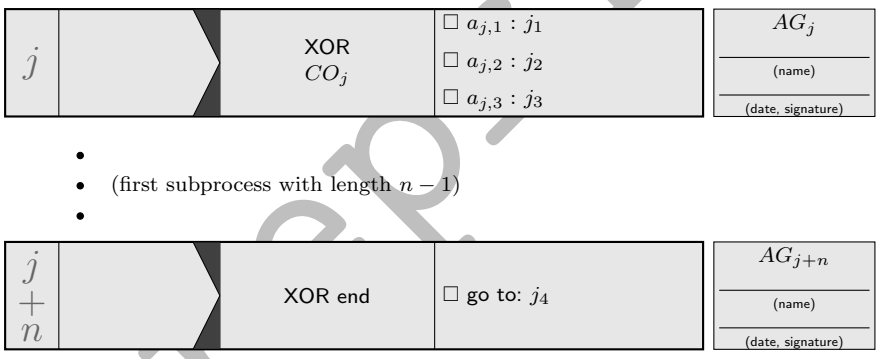

Fig. 16: Representation of the exclusive gateway of Fig. 14 and a jump instruction after the first subprocess transferring the performer to the task with checklist number j_4 after the join gateway of Fig. 15.

5.3.2 Transformation of Parallel Gateways

The transformation methods of parallel gateways correspond to that ones already listed in Section 4.2.2 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 the previous section. Of course, a mixture of several transformation methods is possible, too (see, e.g., Fig. 8 and its explanation). Due to the demanded block-structure there are always pairs of split and join gateways.

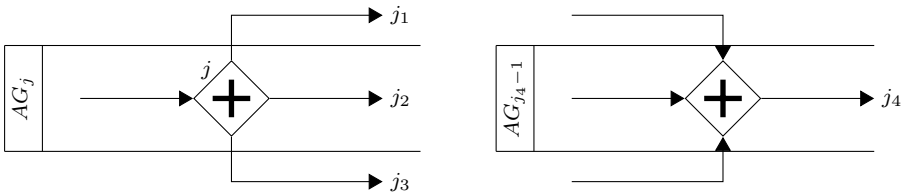

Fig. 17: Parallel split gateway p_j^c and parallel join gateway $p_{j_4-1}^c$.

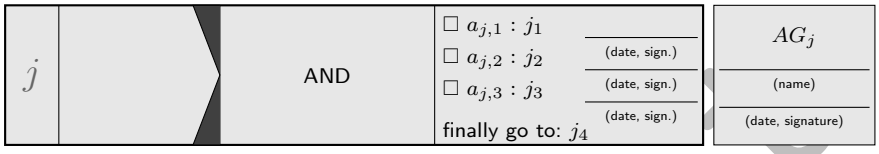

Fig. 18: Checklist representation of the parallel split in the dynamic sequential way without the corresponding jump instructions that refer to agent AG_j of point No. j after every subbranch.

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 imposed by parallelism is completely ignored.

Dynamic Sequential and Postbox Transformation This transformation results in a checklist as described in Section 4.2.2, 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 in a sequential way in the checklist. At the end of each branch a jump to p_j^c, realized with a simple control point like in Fig. 16 except that the go-to-instruction points back to point j instead of j_4, is necessary and 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 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 Section 4.2.2, 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 sub-checklists, $g_{j,1},\ldots,g_{j,k} = 1$ (i.e., the first points of the subchecklists) and $c_j = 1$ which means that signatures for all returning subchecklists are needed (see Section 4.2.2 and Section 5.2).

5.3.3 Transformation of Inclusive Gateways

The transformation of inclusive gateways can be done similar 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 Section 4.2.3). The only difference is, that in p_j^c we have CO_j and $a_{j,1},\ldots,a_{j,k}$ like in the exclusive gateway transformation, i.e., the condition/question and the answers have to be taken over from the process model.

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. 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.

5.4.1 Direct Transformation of Events

Some events, like signal, escalation, or compensation events, or milestones (untyped catching intermediate events) can be transformed like activities, that means 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.

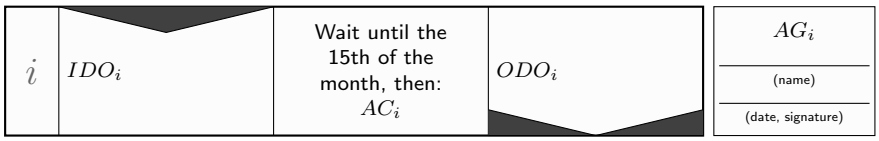

Fig. 19: Indirect representation of an intermediate catching timer event in the event-following activity number i.

5.4.3 Ignored Events

Other events, like the start event, can be ignored, that means they have no respresentation 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. So, 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 Section 4), it can be handled very flexible by the agents. This does not mean, that the agents can do what they want during the execution, but they have a certain freedom which is not unrestrictedly given when processing with 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. [8]). What if a certain treatment has been forgotten because of its rareness, 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 $a_{j,l+1} = \mathit{other}$ and $g_{j,l+1}$ 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 to get 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 indicated above that also in traditional WfMSs flexibility can be achieved. In literature, when talking about flexibility it is distiguished between different types of flexibility, like flexibility by underspec-

ification, flexibility by change, or flexibility by deviation (see [20]). However, as stated in [24], WfMSs usually do not support all of the different types of flexibility. 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 Section 7.1.

6 Implementation

In the following section we describe the implementation of the Process Checklist approach. Therefore, we give insights in the underlying meta-model as well as the model transformation procedures. Furthermore, we describe a XML-based serialization method of checklists.

6.1 The Checklist Meta-Model

The checklist meta-model as a UML class diagram is shown in Fig. 20 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 checklist, a new instance of the meta-model is created. A checklist consists of several *ChecklistElements* that can either be *OperatingPoints* or *ControlPoints*. An OperatingPoint can have several ingoing as well as outgoing *DataObjects* and is performed by exactly one *ResponsibleRole*. A ControlPoint has several *GoTo*-instructions and has also exactly one ResponsibleRole. A GoTo-instruction refers to a ChecklistElement again. The confirmation attribute of ControlPoint corresponds to variable c in the definition of checklist vectors (Definition 1) that 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 in a few seconds.

In a first step, the user selects the BPMN-XML file which should be transformed from a file dialog. Subsequently, the selected file is parsed by the application. As a result, an instantiation of the (simplyfied) BPMN meta-model is generated. Note, that for this approach we are only considering the basic BPMN elements as described in Section 5, especially in Definition 2. Afterwards, the user has to choose the transformation method of possibly occurring parallel gateways, i.e., whether a static sequential, a dynamic sequential

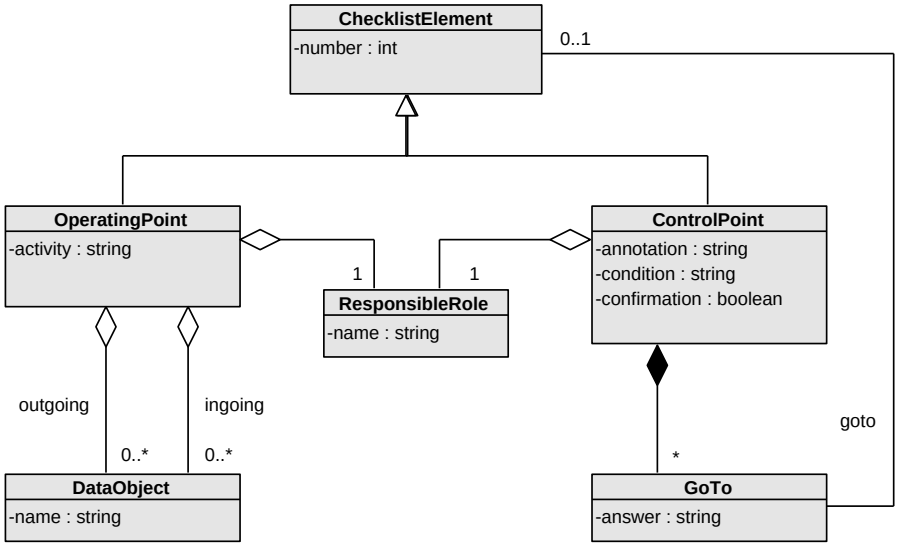

Fig. 20: Meta-model of the Process Checklist.

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 as shown in Fig. 20.

The different checklist model elements can finally be read and visualized. Of course, there is no fixed checklist visualization. However, the visualization 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. Therefore, 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 that are called with different parameters. Shapes, fonts, and colors are defined within these command definitions and can therefore be adjusted to the users' needs. A operating point *perform exam correction* with the running number 5 that consumes a data object *exam unmarked*, produces a data object *examed 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 from the system environment. Furthermore, representational details can be adjusted directly by users and independent from a specific implementation.

6.3 Checklist Serialization

In order to be able to adequatly save and share generated checklists, additionally, a possibility to serialize checklists to XML[3] is provided. The following listing shows an excerpt of the generated XML code of the example checklist vector of Fig. 5.

```
<checklist process='Subscribing for an exam'>
      ...
      <controlpoint no='2' annotation='' condition='Exam type?'
       confirmation='0'>
            <gotolist>
                  <goto answer='written' gotoNo='3' />
                  <goto answer='oral' gotoNo='9' />
            </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>
```

Out of this data stream a checklist based on the meta-model of Fig. 20 can be reconstructed independently of the underlying architecture.

7 Evaluation and Analysis

In this section, two case studies, one in the academic field and one within a bank, are presented. 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 Section 7.1. The other criteria, namely parallelism, length, comprehensibility, orientation, and reliability, are self-descriptive. Sections 7.2 and 7.3 represent the case studies.

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 evaluation 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

[3] Complete example checklist XML-files as well as a XML Schema definition can be found at the project website. See *checklists.kppq.de* for more information.

second case study again addresses the user's comprehensibility and additionally the acceptance of the user which is an important factor when introducing new procedures in companies. Also in the second evaluation, we want to get information whether Process Checklists are needed at all.

The first evaluation is primarily a comparison of process execution through checklists and through BPMN models and a comparison of the different transformation methods shown in Section 5. Basically, there are differnt forms of deploying graphical BPMN models. A very simple form is to publish BPMN models – for example on the intranet of a company. Process participants now are obliged to take these models into account when working on processes. Although this enactment is very cheap, it lacks liability. 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. Small and medium companies cannot and 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 liability. In the university where we carried out the first case study, a WfMS ws 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 an internal 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 Flexibility of Checklists

In [26], 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 [26]." 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 Section 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 [26]." This type of flexibility is fast and easily achieved simply by changing the checklist, e.g., inserting an additional task or 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.

"*Flexibility by underspecification* is the ability to execute an incomplete process model at run-time, i.e., one which 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 [26]." 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 utilized data at run-time. See Section 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 [26]." 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 checklist pass or introducing electronical 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 Section 4.2.1 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 flexbility types proposed in [26]. Flexibility by design is a type that is available in virtually all process modelling languages. Admittedly, this flexibility type is easier to realize for graphical process modelling languages than for checklists. Flexibility by deviation and flexibility by underspecification are feasible quite simply as changing single process instances just means to "rewrite"

the checklist with a pen. Flexibility by change, however, is a problem and difficult to achieve. This is, regarding only the flexibility criterion, why we suggest to use 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 infeasible. They are not recommended in fields where, for example, the processes strongly depend on normative or legal regulations and these regulations are changed quite often so that the underlying process has to be changed, too. 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.

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 Section 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 modeled to get more reliable results.

We modeled the example "subscribing for an exam" as a graphical BPMN process (see Fig. 26) and as a Process Checklist. Parts of the checklist are represented in Fig. 1. The corresponding checklist vector as basis for the graphical checklist is presented in Fig. 5. The first process of the academic domain case study included 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. 27). Additionally, two different checklists to examine the differences presented in Section 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 three support tools for both processes.

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

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 organizational perspective) and their order (the control-flow perspective) as prescribed by the support tool can't be changed. A medium value means that the order of activities can be customized 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 Section 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 Section 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.

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 initialization 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 in [6]. Parallelism is high if real parallelism is possible, i.e., a fully independent execution of several subprocesses. For the "Subscribing for an exam" process model no parallelism values could be distributed as the are no parallel gateways in the BPMN process model.

Length of the process execution support tools 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 shall only give 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. Table 1 shows the results of the objective evaluation criteria for the two processes "Subscribing for an exam" and "Applying for a business trip".

7.2.2 Subjective Criteria

As mentioned in the introduction of Section 7.2, the subjective evaluation criteria are comprehensibility, orientation and reliability. The 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 tools: one BPMN model and one checklist for the first process and one BPMN model and two checklists for the second process. About the half of them was 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[4] allow a good interpretation of the results for the three criteria.

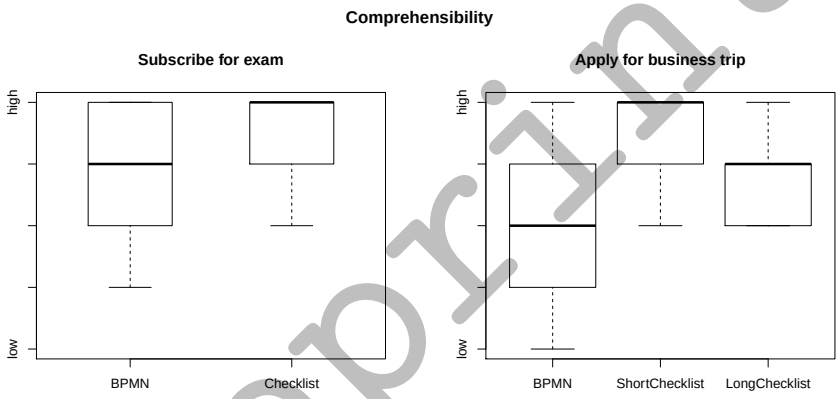

Fig. 21: Boxplot evaluation of the comprehensibility for the two processes "Subscribe for exam" (BPMN model and checklist) and "Apply for business trip" (BPMN model, a short checklist, and a long checklist).

In Fig. 21 the boxplots for comprehensibility of the different process execution 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 execution 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. 21, the scattering for both BPMN models is greater than that of the checklists. This could occur from the fact, that 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.

[4] For more information about boxplots see [10], Section 2.2, and [29].

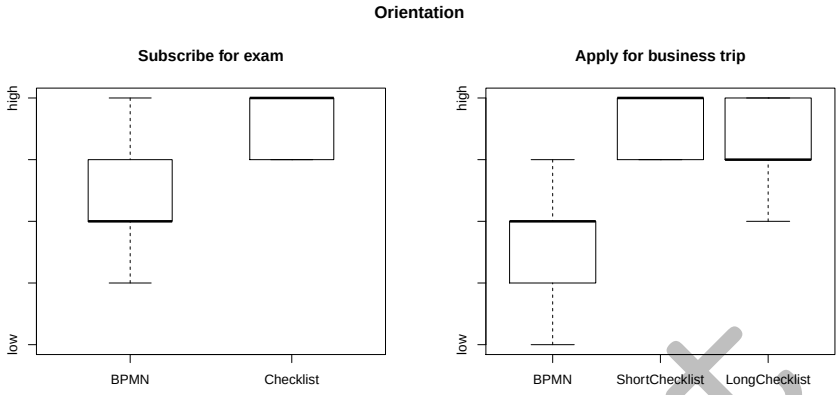

Fig. 22: Boxplot evaluation of the orientation for the two processes "Subscribe for exam" (BPMN model and checklist) and "Apply for business trip" (BPMN model, a short checklist and a long checklist).

The second subjective evaluation criterion is orientation, which asks for knowing the own position during the process execution and the steps that have to be performed next. The possible answers reached 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. 22.

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, which means better, for the checklists than for the BPMN model. As it can be seen, the short checklist for the business trip process has a better orientation value than the long checklist. This could be caused by 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 sequential task chains which allow for easier orientation. Moreover, one conspicuity 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. 23, 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 simply means the opposite, i.e., the execution is not traceable and it is not clear, who has carried out which tasks. It sticks out that the median values for the BPMN support tool are quite low and for the checklists have even increased compared to 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

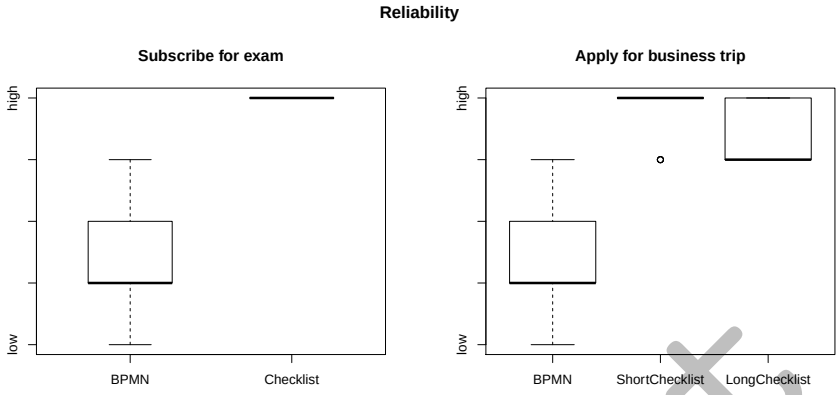

Fig. 23: Boxplot evaluation of the reliability for the two processes "Subscribe for exam" (BPMN model and checklist) and "Apply for business trip" (BPMN model, a short checklist, and a long checklist).

BPMN model again vary very much, but at a rather low level. So, 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.

Additionally, we want to remark that even if graphical process models are sometimes the only available process support tools in (mostly small and middle-sized) 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. And especially for that companies that already have graphically modelled processes the enactment of Process Checklists is inexpensive and could lead to evaluation improvements like they are shown above.

7.3 Bank Case Study

The second case study was performed together with *Sparkasse Bamberg*, a German savings bank with 864 employees and a balance sheet total of EUR 3.5189 billion (both in 2014) [28]. One process about how to handle 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 comprehen-

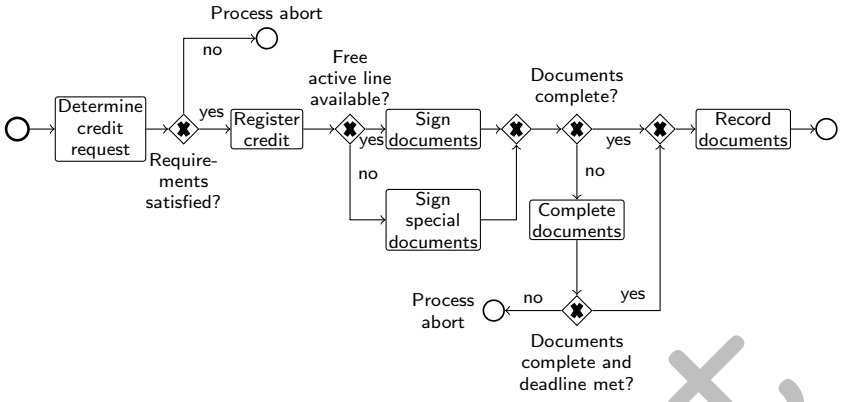

Fig. 24: BPMN model of an overdraft facility process in a shortened version without data and human resources.

sion of the process without having to introduce another modelling language and to make use of the transformation rules stated in Section 5 for generating the checklist. The BPMN model has been validated by several employees including the future CEO of the bank. A shortened, translated version of the overdraft facility process is shown in Fig. 24.

Afterwards, the BPMN process model was transformed to a Process Checklist according to the rules from Section 5. This serialization was done in merely about one hour. The checklist bundle, i.e., the Process Checklist and the cover sheet, was then handed over to the employees of the savings bank to run the process with help of the Process Checklist instead of processing the overdraft facility with the instructions provided by the internally available process model. A written manual with 2.5 pages about how to use the checklist was the only instruction the 31 employees that took part in the case study received. Like 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 taken out 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. 25 and summarized 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 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

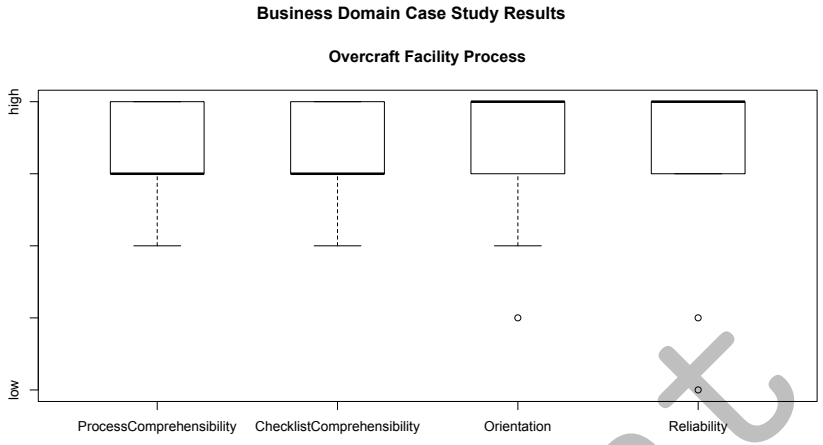

Fig. 25: Results of the financial domain case study.

(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 rated quite well with a few exceptions. At leat 50% assigned the highest value for reliability and at least 75% assigned at least the second highest value.

These results of the subjective evaluation criteria basically seem to affirm our approach but the most valuable outcomes are the (positive and negative) critics and the proposals for improvement of the Process Checklist approach. Basically, the critics and proposals could be divided into two categories: statements concerning the checklist and statements concerning the overcraft facility process. We show no further interest in the latter mentioned 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 critics, negative critics and improvement proposals and are classified into several criteria. Issues declared as frequently are mentioned three or more times in the evaluation.

Positive Critics 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.* Further more, *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 Critics 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 can be made. First of all, the cover sheet is redundant in that 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 proceeded sequentially, i.e., if jumps (forward or backward) are performed and so the point to be performed next is either not recognizable or only recognizable with a certain amount of reproducing efforts. In the following improvements section a proposal for optimizing 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 where Process Checklist 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 anyway. Another improvement proposal relates to the extention 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 *summarize 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 summarized 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 that 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 operatin 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 explicitely wanted to examine and test paper-based Process Checklists. However, the structure of Process Checklists as introduced in Section 2 may likely be transferred to electronic process execution support tools. 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 shall be given, highlighting also the restrictions of the proposed Process Checklist approach, as well as further issues for future work that could be spent on this topic.

8 Conclusion, Limitations, and Future Work

In order to diminish the dependency from IT-based process management systems, the work at hand proposed an alternative way of supporting 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 handbook. The Process Checklist is handed over during process execution from process participant to process participant. Successful task accomplishments are recorded through 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 serialized 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 general structure of Process Checklists as well as a transformation algorithm of basic BPMN process model elements to Process Checklists. Furthermore, we described implementation details by giving a concrete checklist meta-model as well as a XML-based serialization 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 showed that Process Checklists serve as a feasible process execution support and are highly accepted by process participants.

In contrast to the advantages over IT-based process management systems as mentioned before, paper-based checklists can also have disadvantages com-

pared to traditional systems. 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 administrational environments. Moreover, one disadvantage may occur due to human failure as the Process Checklist, i.e, one process instance, simply may get lost. Recovering from such an incident can be accomplished with the help of receipts but this task may be laborious. However, the problem of losing documents is not only a topic 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 serialization of the process, the checklist representation has of course problems when dealing with parallelism. Here, process modelers have to choose a suitable transformation method as described in Section 5. The presented case studies focused on a first evaluation in the fields of university processes and banking processes. Here, we got useful experiences 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 base 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, e.g., Declare [24] or DPIL [32], that already contain a high degree of flexible process execution but need, due to this extreme flexibility, appropriate navigation and support tools.

Acknowledgements The authors gratefully acknowledge the superb collaboration with *Sparkasse Bamberg*. Especially, the authors wish to thank Mr. Stephan Kirchner, board member and future CEO of *Sparkasse Bamberg*.
The presented work is developed and used in the project "Kompetenzzentrum für praktisches Prozess- und Qualitätsmanagement", which is funded by "Europäischer Fonds für regionale Entwicklung (EFRE)".
The work of Michael H. Baumann is supported by a scholarship of "Hanns-Seidel-Stiftung e.V. (HSS)" which is funded by "Bundesministerium für Bildung und Forschung (BMBF)".

References

1. van der Aalst, W.M., Weske, M., Grnbauer, D.: Case handling: a new paradigm for business process support. Data & Knowledge Engineering **53**(2), 129 – 162 (2005)
2. Baumann, M., Baumann, M.H., Schönig, S., Jablonski, S.: Enhancing feasibility of human-driven processes by transforming process models to process checklists. In: I. Bider, K. Gaaloul, J. Krogstie, S. Nurcan, H. Proper, R. Schmidt, P. Soffer (eds.)

Enterprise, Business-Process and Information Systems Modeling, *LNBIP*, vol. 175, pp. 124–138. Springer Berlin Heidelberg (2014)
3. Boehm, B.: Software risk management: principles and practices. Software, IEEE **8**(1), 32–41 (1991)
4. Briere, J.: Trauma symptom checklist for children. Odessa, FL: Psychological Assessment Resources pp. 00,253–8 (1996)
5. Condon, C.: The computer wont let me: Cooperation, conflict and the ownership of information. In: S. Easterbrook (ed.) CSCW: Cooperation or Conflict?, CSCW, pp. 171–185. Springer London (1993)
6. Degani, A., Wiener, E.L.: Human factors of flight-deck checklists: the normal checklist. NASA Contractor Report 177549 (1991)
7. Derogatis, L.R., Lipman, R.S., Rickels, K., Uhlenhuth, E.H., Covi, L.: The hopkins symptom checklist (hscl): A self-report symptom inventory. Behavioral Science **19**(1), 1–15 (1974)
8. Ely, J.W., Graber, M.L., Croskerry, P.: Checklists to reduce diagnostic errors. Academic Medicine **86**(3), 307–313 (2011)
9. Faerber, M., Jablonski, S., Schneider, T.: A comprehensive modeling language for clinical processes. In: ECEH, pp. 77–88. Citeseer (2007)
10. Fahrmeir, L., Künstler, R., Pigeot, I., Tutz, G.: Statistik – Der Weg zur Datenanalyse. Springer-Verlag (2007)
11. Hales, B.M., Pronovost, P.J.: The checklist – a tool for error management and performance improvement. Journal of Critical Care **21**(3), 231 – 235 (2006)
12. Hartel, M.C., Chou, S.C.: Electronic checklist system. United States Patent (Patent Number 5 454 074) (1995)
13. Hauser, R., Friess, M., Kuster, J., Vanhatalo, J.: Combining analysis of unstructured workflows with transformation to structured workflows. In: Enterprise Distributed Object Computing Conference, pp. 129–140 (2006)
14. Jablonski, S.: Do we really know how to support processes? considerations and reconstruction. In: G. Engels, C. Lewerentz, W. Schfer, A. Schrr, B. Westfechtel (eds.) Graph Transformations and Model-Driven Engineering, *LNCS*, vol. 5765, pp. 393–410. Springer Berlin Heidelberg (2010)
15. Jablonski, S., Bussler, C.: Workflow management: modeling concepts, architecture and implementation. International Thomson Computer Press (1996)
16. Kerzner, H.R.: Project management: a systems approach to planning, scheduling, and controlling. John Wiley & Sons (2013)
17. Koehler, J., Hauser, R., Kster, J., Ryndina, K., Vanhatalo, J., Wahler, M.: The role of visual modeling and model transformations in business-driven development. Electronic Notes in Theoretical Computer Science **211**(0), 5 – 15 (2008). Proceedings of the Fifth International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT 2006)
18. Kopp, O., Martin, D., Wutke, D., Leymann, F.: The difference between graph-based and block-structured business process modelling languages. Enterprise Modelling and Information Systems Architecture **4**(1), 3–13 (2009)
19. Luff, P., Heath, C., Greatbatch, D.: Tasks-in-interaction: Paper and screen based documentation in collaborative activity. In: Computer-supported Cooperative Work, CSCW, pp. 163–170. ACM (1992)
20. Mans, R., van der Aalst, W., Russell, N., Bakker, P.: Flexibility schemes for workflow management systems. In: D. Ardagna, M. Mecella, J. Yang (eds.) Business Process Management Workshops, *LNBIP*, vol. 17, pp. 361–372. Springer Berlin Heidelberg (2009)
21. Melenovsky, M.J.: Business process management's success hinges on business-led initiatives. Gartner Research, Stamford, CT **July**, 1–6 (2005)
22. zur Muehlen, M., Recker, J.: How much language is enough? theoretical and practical use of the business process modeling notation. In: Z. Bellahsne, M. Lonard (eds.) Advanced Information Systems Engineering, *LNCS*, vol. 5074, pp. 465–479. Springer Berlin Heidelberg (2008)
23. Object_Management_Group_Inc.: Business process model and notation (bpmn) version 2.0 (2011). URL `http://www.omg.org/spec/BPMN/2.0`
24. Pešić, M.M.: Constraint-based workflow management: Shifting control to users. Ph.D. thesis, Technische Universiteit Eindhoven (2008)

25. Reichert, M., Weber, B.: Enabling flexibility in process-aware information systems: challenges, methods, technologies. Springer Science & Business Media (2012)
26. Schonenberg, H., Mans, R., Russell, N., Mulyar, N., van der Aalst, W.: Process flexibility: A survey of contemporary approaches. In: J. Dietz, A. Albani, J. Barjis (eds.) Advances in Enterprise Engineering I, *LNBIP*, vol. 10, pp. 16–30. Springer Berlin Heidelberg (2008)
27. Seitz, M., Schönig, S., Jablonski, S.: A framework for reasonable support of process compliance management. In: W. Abramowicz, A. Kokkinaki (eds.) Business Information Systems Workshops, *LNBIP*, vol. 183, pp. 131–144. Springer International Publishing (2014)
28. Sparkasse_Bamberg: Report 2014 (2015-04-06). URL `https://www.sparkasse-bamberg.de/pdf/preise_leistungen/2014_report_spk_bamberg.pdf`
29. The_R_Foundation: R: The r project for statistical computing (2015-04-06). URL `http://www.r-project.org/`
30. Wolff, A.M., Taylor, S.A., McCabe, J.F.: Using checklists and reminders in clinical pathways to improve hospital inpatient care. Medical Journal of Australia **181**, 428–431 (2004)
31. Zairi, M.: Business process management: a boundaryless approach to modern competitiveness. Business Process Management Journal **3**(1), 64–80 (1997)
32. Zeising, M., Schönig, S., Jablonski, S.: Towards a Common Platform for the Support of Routine and Agile Business Processes. In: Proceedings of the 10th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing (2014)

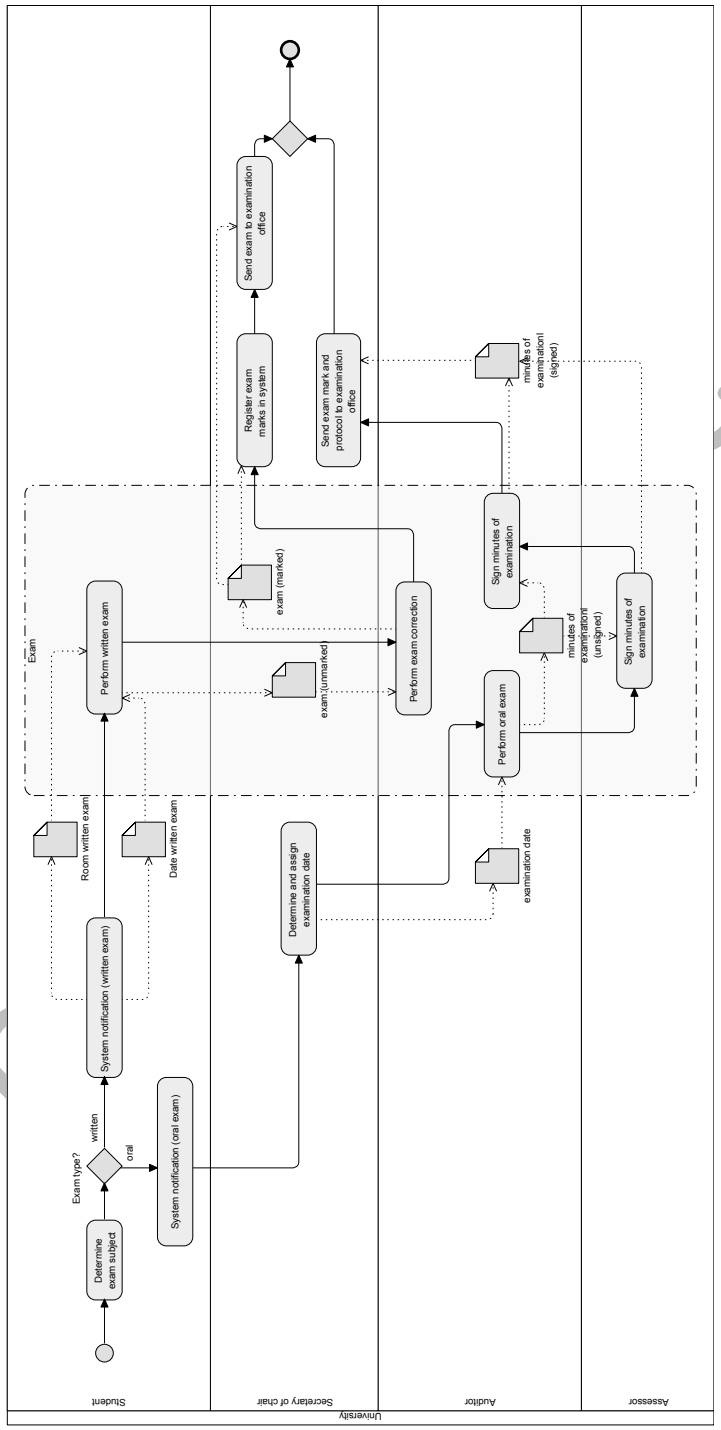

Fig. 26: BPMN model for the process “Subscribing for an exam”.

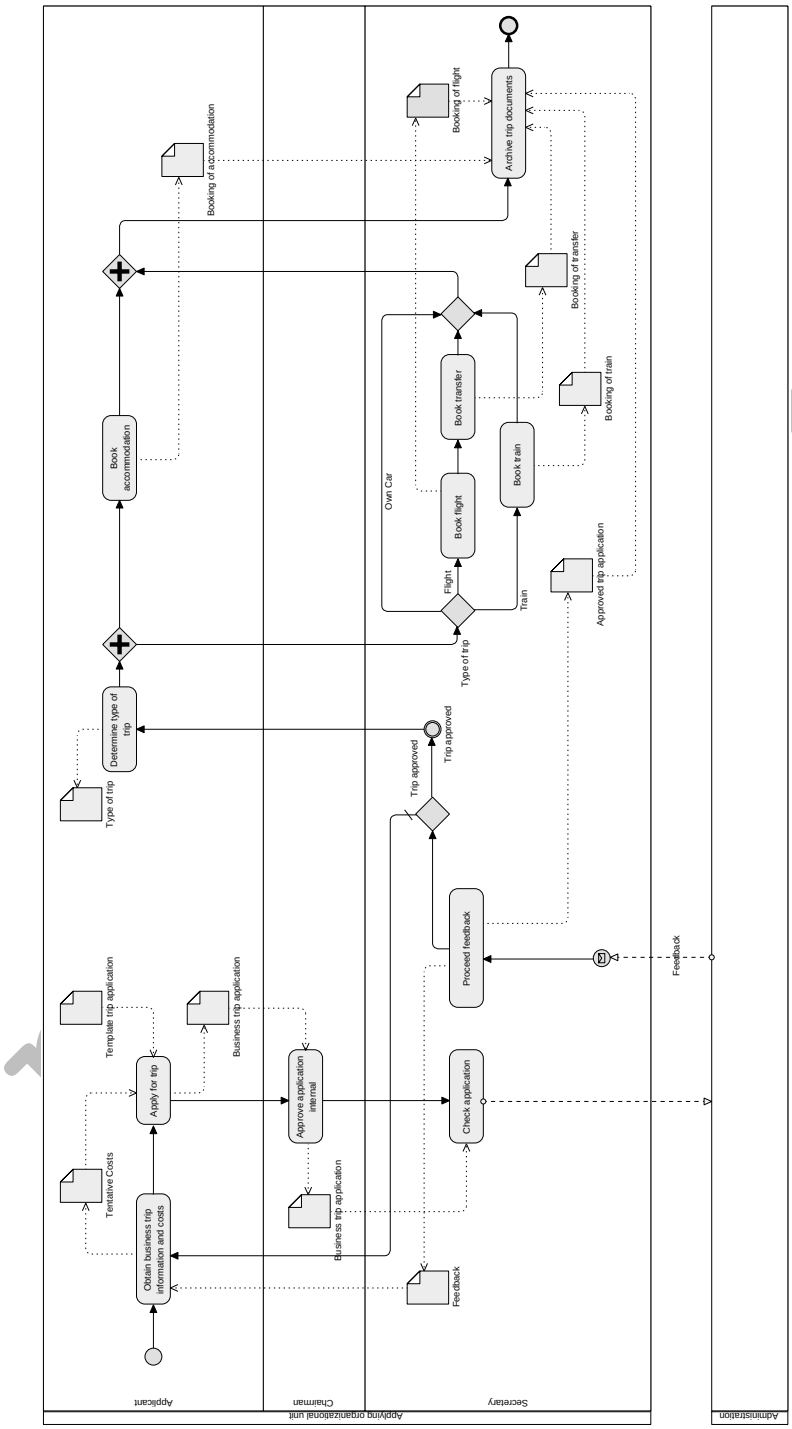

Fig. 27: BPMN model for the process “Applying for a business trip”.