Enhancing Feasibility of Human-Driven Processes by Transforming Process Models to Process Checklists

Michaela Baumann, Michael Heinrich Baumann, Stefan Schönig, and Stefan Jablonski

University of Bayreuth, Germany
{michaela.baumann,michael.baumann, stefan.schoenig,stefan.jablonski}@uni-bayreuth.de

Abstract. In traditional approaches business processes are executed on top of IT-based Workflow-Management Systems (WfMS). The key benefits of the application of a WfMS are task coordination, step-by-step guidance through process execution and traceability supporting compliance issues. However, when dealing with human-driven workflows, conventional WfMS turn out to be too restrictive. Especially, the only way to handle exceptions is to bypass the system. If users are forced to bypass WfMS frequently, the system is more a liability than an asset. 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. We introduce the so-called process checklist representation of process models where processes are described as a paper-based step-by-step instruction handbook.

Keywords: process modelling, process checklists, paper-based process execution.

1 Introduction

Since approximately 20 years process management is regarded as an innovative technology both for the description of complex applications and for supporting their execution [1]. In traditional approaches business processes are executed on top of IT-based Workflow-Management Systems (WfMS) [2]. The key benefits of the application of a WfMS are task coordination, step-by-step guidance through process execution and traceability supporting compliance issues [3]. However, when dealing with human-driven workflows that heavily depend on dynamic human decisions, conventional WfMS turn out to be too restrictive [4]. 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 WfMS frequently, the system is more a liability than an asset [4]. 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 [6].

Furthermore, the introduction of a WfMS is regarded as a huge, cost-intensive project [7]. Many organizations cannot afford to introduce such a system therefore. 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 and authorities. We introduce the so-called process checklist representation of process models. Here, 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 agents. In principle the most important statement is that at the end of the process all 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 extremely flexible process execution. Besides, the process checklist also supports the key benefits of traditional WfMS. 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 BPMN process model elements [8] to process checklists. Fig. 1 shows a comparison of traditional IT-based process execution and the paper-based approach provided by the work at hand.

$$\begin{tikzcd}
 & \text{model} & \text{execution} \\
\text{IT-based} & \text{BPMN} \arrow[r] \arrow[d, "\textit{trans-formation}"] & \text{WfMS} \\
\text{paper-based} & \text{checklist vector} \arrow[r] & \text{graphical checklist}
\end{tikzcd}$$

Fig. 1. Schematic approach of distancing from IT-based process management systems

2 Background and Related Work

A checklist is a list of items required, things to be done, or points to be considered, used as a reminder [9]. Checklists are generally seen as a suitable means for error management and performance improvement in highly complex scenarios like clinical workflows [10]. Therefore, we propose to define a generic method for

transforming general process models to the checklist representation. The problem of transforming a model drawn in one business process modeling notation into another notation has been examined in different papers, e.g., [11], [12]. However, to the best of the authors' knowledge, the transformation of process models to a checklist representation has not been discussed so far.
Before specifying the transformation of process models into checklists, we have to determine how suitable process models should look like and what elements a checklist consists of. These specifications are necessary to give concrete mapping rules. For process models, only basic elements of the Business Process Modeling Notation are allowed, as [13] shows this is enough in most cases and as the paper at hand has to be seen as a first approach to this topic.

Definition 1 (Process model). *A process model is defined according to BPMN 2.0 (see e.g. in [8]) 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*

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. Which forms of activities, events and gateways can be covered with our transformation rules will become apparent when it comes to the concrete transformation of process models into checklists. We specify a checklist as follows.

Definition 2 (Checklist vector). *A checklist is a vector $\mathcal{C} = (p_1^t, p_2^t, \dots, p_n^t)$, $n \in \mathbb{N}$, $t \in \{o, c\}$ with two different kinds of components:*

$$p_i^o = (ID_i, AC_i, OD_i, AG_i)$$

with ID_i, AC_i, OD_i, AG_i being strings and

$$p_j^c = (AN_j, CO_j, GT_j, AG_j)$$

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

$$GT_j = (s_j, a_{j,1}, g_{j,1}, a_{j,2}, g_{j,2}, \dots, a_{j,k}, g_{j,k})$$

with $k \in \mathbb{N}$, strings $a_{j,l}$, integers (or NULL) $g_{j,l} \in \{1, \dots, n\} \cup \{NULL\}$, $l = 1, \dots, k$, and $s_j \in \{0, 1\}$.

This definition uses a lot of different variables that need some explanation: The first component of a checklist vector, p^o, is called operating point. It contains information about incoming data objects (ID), the activity (AC) which may be an activity in the literal sense of BPMN or an event, outgoing data objects (OD), and the performing agent (AG). An operating point gives more or less conrete instructions to the respective agent about what he has to do. The other component of a checklist vector, p^c, is called control point. In general, a control point is a transformed gateway, therefore it contains information about the condition (CO) which may also be empty if it corresponds to a parallel gateway, and the responsible agent (AG as in p^o). AN is a component kept free for special annotations (we will see examples later) and GT is a vector with one boolean component s and k pairs of string (a) and integer (g) components. g refers to other components of $\mathcal{C}$ and is therefore element of $\{1, \ldots, n\}$ or $NULL$. With this formal definition of a checklist, the checklist vector, it is already possible to give concrete mapping instructions as listed in the next section. Before we turn towards this subject, we want to give the reader a visual impression of how the two components p^o and p^c may be illustrated on a graphical checklist in Fig. 2 and Fig. 3. The components of vector GT_j are shown in Fig. 4.

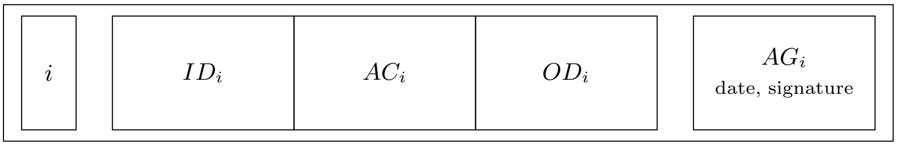

Fig. 2. Visualization of p_i^o which means the i-th component of $\mathcal{C}$ is an operating point

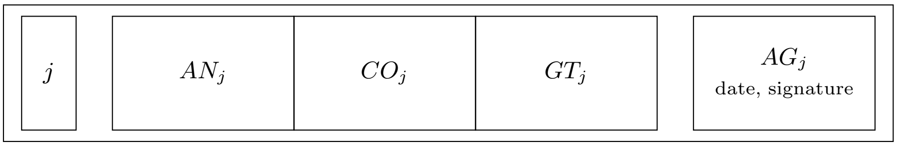

Fig. 3. Visualization of p_j^c which means the j-th component of $\mathcal{C}$ is a control point

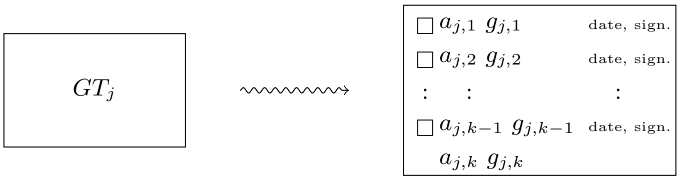
GT_j
$\square\ a_{j,1}\ g_{j,1}$ date, sign.
$\square\ a_{j,2}\ g_{j,2}$ date, sign.
$\square\ a_{j,k-1}\ g_{j,k-1}$ date, sign.
$a_{j,k}\ g_{j,k}$

Fig. 4. Visualization of GT_j. Entries date and signature in the third column only appear, if $s_j = 1$. The k-th row never has a square in the first column nor a date and signature. In fact, the k-th row may be empty, i.e., $a_{j,k} =$ “” and $g_{j,k} = NULL$.

In which way these checklist components are filled with information given by the process model and how the resulting operating and control points are represented in the graphical checklist is explained in the next two sections.

3 Transformation of Process Model Elements

This section focuses on generating a checklist, that means it is explained, in which way the single elements of the (BPMN) process model are transferred into either operating points or control points. These steps are basically performed in a simply algorithmic way, except for parallel gateways.

3.1 Transformation of Activities

Activities are transformed straight into operating points p^o. Their description is 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, e.g., be a single person is mapped onto the field AG. An example of an activity with documents and participant is given in Fig. 5.

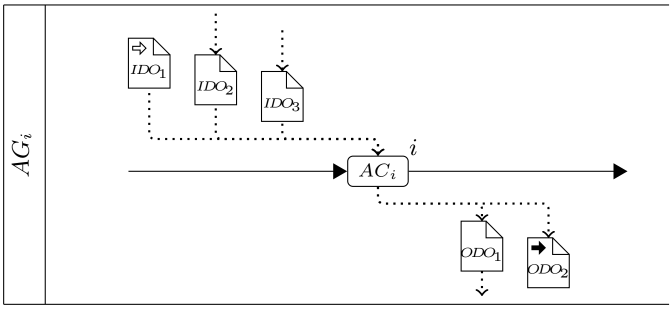

Fig. 5. Exemplary excerpt from a process model with labels according to an operating point p^o_i. $ID_i = IDO_1.IDO_2.IDO_3$ and $OD_i = ODO_1.ODO_2$.

3.2 Transformation of Subprocesses

Occurring subprocesses, marked with a symbol as seen in Fig. 6, may be taken into a checklist in different ways:

1. Include the complete subprocess (comparatively long, but correct checklist)
2. Generate a new checklist for each subprocess (insertion of two operating points into the original checklist ist necessary: one with work instructions for printing/passing on the new checklist, one with work instructions for waiting for the finished subprocess, subprocess checklist as incoming data)

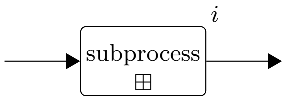

Fig. 6. Symbol for a subprocess in BPMN 2.0

3.3 Transformation of Gateways

Transformation of Exclusive Gateways. An exclusive split gateway (Fig. 7) has to be transformed into a control point in which the decision question and the possible answers with the respective "go to"-numbers ($g_{j,1}, g_{j,2}$ and $g_{j,3}$ in Fig. 7) are mentioned. If there is a exclusive join gateway (Fig. 8) too, at the end of each branch of the respective splitting gateway a jump instruction to the next point in the checklist after the join gateway ($g_{j,4}$ in Fig. 8) must be inserted, except the next point following a branch is the point following the join gateway. The execution of an exclusive gateway may cause problems if at least one "go to"-number is in the past, but this problem will be solved in the next section.

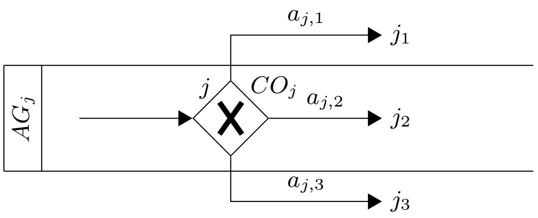

Fig. 7. Exclusive split gateway with question CO_j and possible answers $a_{j,1}, a_{j,2}, a_{j,3}$. p_j^c: AN_j may be used for data. $g_{j,k} = j_k$, $k = 1, 2, 3$, $a_{j,4} =$ "" and $g_{j,4} = NULL$.

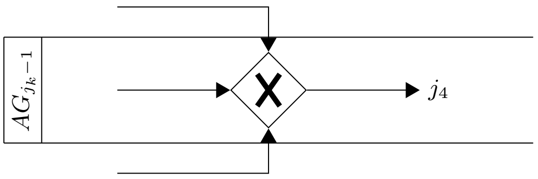

Fig. 8. Exclusive join gateway that does not have to exist if the outgoing branches of the exclusive split gateway end with terminal events. $p_{j_k-1}^c$: $AN_{j_k-1} =$"", $CO_{j_k-1} =$ "*XOR end*", $s_{j_k-1} = 0$, $a_{j_k-1} =$ "*goto :*", $g_{j_k-1} = j_4$, $k = 2, 3$.

Transformation of Parallel Gateways. There are several ways of transforming parallel gateways into a checklist whereby all of them have different advantages and disadvantages. Some of these possibilities are listed below. Note, that a mixture of these transformation possibilities is also conceivable.

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 (Fig. 9), and brings them into an arbitrary order. The gateway itself is not mapped to the checklist.

Dynamic Sequential or Postbox Transformation A parallel split will be transformed to a control point p_j^c. The parallel branches in 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, is necessary and in p_j^c the number of the point following the respective parallel join has to be noted. There are different ways of executing this parallel split, and some of them correspond to another transformation, but this will be dealt with in the next section.

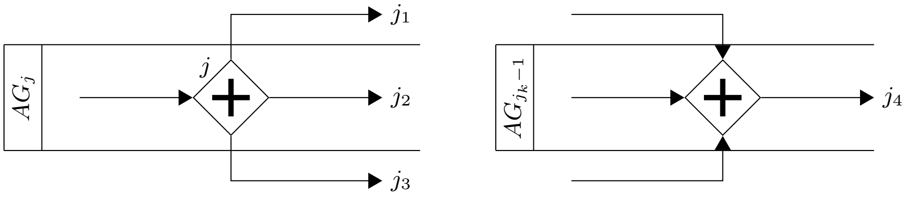

Fig. 9. Parallel split gateway p_j^c: AN_j for annotation, e.g. *DO*s, CO_j = "*AND*", $s_j = 1$, $a_{j,1}, \dots, a_{j,3}$ ="", $g_{j,1} = j_1$, $g_{j,2} = j_2$, $g_{j,3} = j_3$, $a_{j,4}$ ="Finally go to", $g_{j,4} = j_4$, $k = 2, 3, 4$. Parallel join gateway $p_{j_k-1}^c$: AN_{j_k-1} ="", CO_{j_k-1} ="AND end", $s_{j_k-1} = 0$, a_{j_k-1} ="go to", $g_{j_k-1} = j$, $k = 2, 3, 4$.

Parallel Transformation For each parallel branch a checklist is generated and distributed by the agent of the split gateway (see Fig. 9) to the agents of the first process element of the branches. 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 sub-checklists are "Checklist_sub1",...,"Checklist_subk". Of course $a_{j,1}, \dots, a_{j,k}$ have to reference these sub-checklists, $g_{j,1}, \dots, g_{j,k} = NULL$ and $s_j = 1$.

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. The only difference is, that in p_j^c we have CO_j and $a_{j,1}, \dots, a_{j,k}$ like in the exclusive gateway transformation, i.e., the condition/question and the answers have to be taken over from the process model.

3.4 Transformation of Events

Direct Transformation of Events. Some events, like signal events, can be transformed like activities, that means to p_i^o, with $AC_i =$ “” or AC_i is used for transmitting some message.

Indirect Transformation of Events. Most events, like time, condition and message events, are requirements for the next point in the checklist and can be modelled this way. This requirement is written down in AC or AN of the following operating or control point.

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.

4 Enactment of the Graphical Checklist

A graphical checklist contains a cover sheet with name of the checklist (name of the process), timestamp, and a list for writing down the current checklist and the current point, i.e., the next point to be worked on. Furthermore, a graphical checklist consists of at least one checklist as described above (resulting from a checklist vector) with a consecutive number, starting with 1, a receipt book and a list for data objects and maybe data objects (documents). An illustrating example for these components is given in Fig. 10.

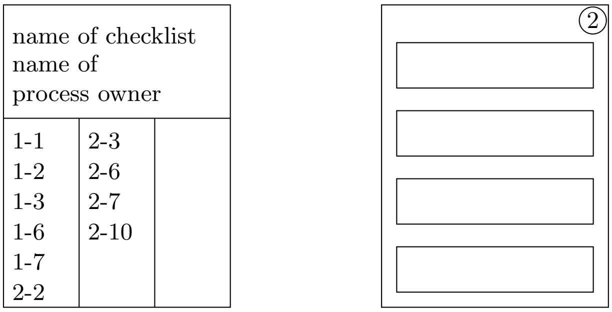

Fig. 10. Cover sheet (left-hand side) with name of the checklist/process, name of process owner and list of the next points to be executed; checklist (right-hand side) with current number in the upper right corner and operating/control points. Obviously, in checklist no. 1 a gateway caused a jump into the past (from point no. 7 to point no. 2)

When starting a process with checklists, the “process owner”, i.e., that 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 number 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 1" and "point 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. This graphical checklist has to be passed to the agent named in point 1, who has to check for completeness, that means especially if all listet documents are handed over, and quit the delivery. The process owner has to archive the signed receipt for later reconstruction if necessary.

Every time 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 1–23 he has to look at point 23 of the current checklist, that has number 1, 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). After execution of the current point he 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, 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 (with possible constraint resulting from a transformed event) 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.

4.2 Execution of Control Points

Execution of Exclusive Gateways. If a control point resulting from an exclusive gateway has to be processed, the agent has to check for the condition or question in field CO. He marks his answer in GT in the box $\square$ in front of the corresponding answer $a_{\cdot,l}$. 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 sceneries 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 $g_{\cdot,l}$ may have been processed already in the past and therefore is signed already. If such a return 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.

Execution of Parallel Gateways.

Static Sequential Transformation If a parallel gateway was transformed in the static sequential way, then it does not appear in the checklist, that means the performing agents do not know, that there has been a gateway in the BPMN process model. All branches are executed in the specific order as chosen by the person who transformed the process model.

Dynamic Sequential Transformation When coming to a control point being the transformation of a parallel gateway with the dynamic sequential method 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. 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, on the right-hand side of the marked box. The branch is processed and at the end there is a control point that refers back to the control point where the decision of the branch was made. So, the agent gets the checklist back (with checking for all documents and quitting again) and signs the chosen branch in GT (that one with the marked box, that has not been signed yet). Then he chooses the next branch to be processed the same way as before. If all branches have been marked and signed, 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.

Postbox Method If parallel gateways are performed with the postbox method, the checklist itself looks the same as transformed according to the dynamic sequential way. The difference is in the execution, as the postbox method allows for parallel processing of the different branches. When the performance of a checklist reaches an AND control node the checklist is posted like an announcement in one place together with all documents (that can be stored in 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. With this method, the documents do not have to be handed over from one point to another. 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 and needs initiative of all agents. But it considers the parallel aspect of parallel gateways.

Parallel Transformation With this method it is also possible to consider simultaneity of the different branches. The agent of the control node prints all required sub-checklists, marks the boxes □ in *GT* if handed over together with needed documents to the respective agents of the first points in the branches, as listed in *GT*, and signs every returning sub-checklist in *GT*. If all sub-checklists have returned, he signs in *AG* and the execution of the control node is finished. As one can imagine, this method is more elaborate, as multiple checklists have to be generated, but it provides a good overview over the process in contrast to the postbox method. We recommend this method if the branches are relatively long, so that the effort of generating more than one checklist is somehow justified.

The mentioned transformation and execution versions are somehow suggestions, clearly many other versions are imaginable and of course different versions can be mixed.

Execution of Inclusive Gateways. Like the transformation of inclusive gateways, the execution of inclusive gateways can again be seen as a mixture of exclusive and parallel gateways. The agent of the corresponding control point has to choose his anwers (mark the boxes □), in contrast to exclusive gateways possibly more than one, and then for the chosen ones he can proceed like with parallel gateways (except for the static sequential method, as this was no possible transformation for inclusive gateways).

All methods mentioned so far require a well-modelled process model, that means for example, that there are no returns out of AND branches, no document is needed in parallel branches without having a copy of it, or that no document is archived if there is the possibility of a return into the past where this document will be needed again. Changes of the underlying process model involve modifications of the checklist for all future process instances. If problems or questions during the execution of one checklist occur, one should confer with the corresponding process owner.

5 Transformation Example

As an illustrating example the process model given in Fig. 11 is transformed into a checklist vector as seen in Table 12.

For transformation of parallel and inclusive gateways the dynamic sequential method was chosen. That is why the documents "exposé" and "course materials" do not have to be copied for the different branches of the corresponding gateway, as they are performed in a sequential order and the document is handed over together with the checklist. For a graphical checklist, one has to put the table view of the checklist into the visually more appealing form as given in Figures 2, 3 and 4, as it is easier to read. Furthermore, the cover sheet has to be added as well as the receipt book and the document list.

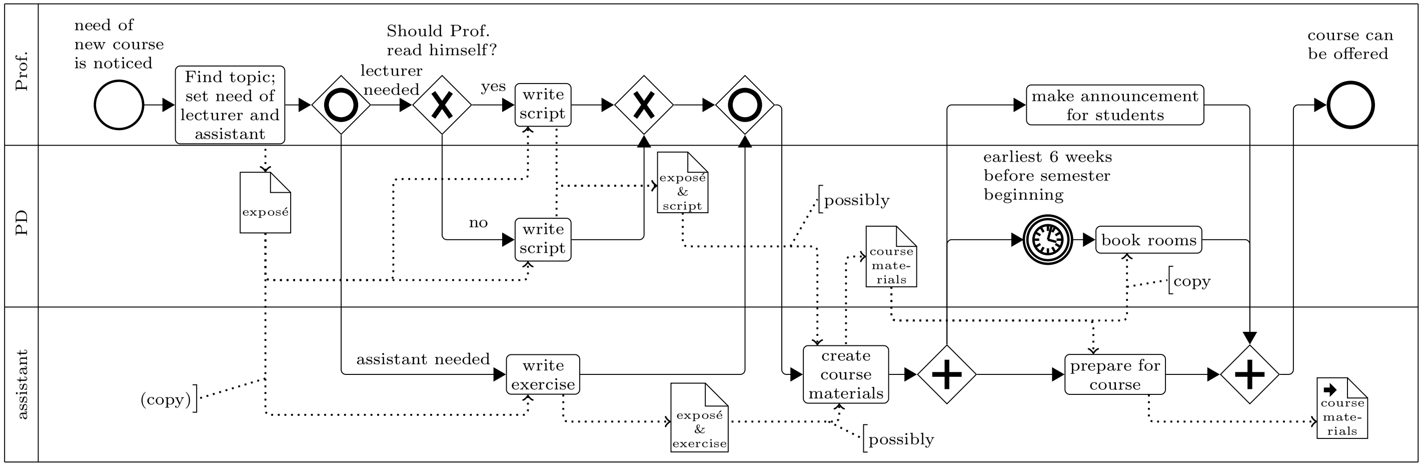

Fig. 11. Exemplary process model: conception of a new course

Table 12. Table view of a checklist vector representating the process model of Fig. 11

No.	type	*ID/AN*	*AC/CO*	*OD/GT*	*AG*
1	o		Find topic; set need of lecturer and assistant	exposé	Prof.
2	c		OR	$s_2 = 1$ $a_{2,1}$ =lecturer needed $g_{2,1} = 3$ $a_{2,2}$ =assistant needed $g_{2,2} = 8$ $a_{2,3}$ =finally go to $g_{2,3} = 10$	Prof.
3	c		XOR Should Prof. read himself?	$s_3 = 0$ $a_{3,1}$ =yes $g_{3,1} = 4$ $a_{3,2}$ =no $g_{3,2} = 6$ $a_{3,3}$ ="" $g_{3,3} = NULL$	Prof.
4	o	exposé	write script	exposé, script	Prof.
5	c		XOR end	$s_5 = 0$ $a_{5,1}$ =go to $g_{5,1} = 7$	Prof.
6	o	exposé	write script	exposé, script	PD
7	c		OR end	$s_7 = 0$ $a_{7,1}$ =go to $g_{7,1} = 2$	Prof.
8	o	exposé	write exercise	exposé, exercise	assistant
9	c		OR end	$s_9 = 0$ $a_{9,1}$ =go to $g_{9,1} = 2$	Prof.
10	o	exposé, script OR exercise	create course materials	course materials	assistant
11	c		AND	$s_{11} = 1$ $a_{11,1}$ =go to $g_{11,1} = 12$ $a_{11,2}$ =go to $g_{11,2} = 14$ $a_{11,3}$ =go to $g_{11,3} = 16$ $a_{11,4}$ =finally go to $g_{11,4} = 18$	assistant
12	o		make announcement for students		Prof.
13	c		AND end	$s_{13} = 0$ $a_{13,1}$ =go to $g_{13,1} = 11$	Prof.
14	o	course materials	earliest 6 weeks before semester beginning: book rooms		PD
15	c		AND end	$s_{15} = 0$ $a_{15,1}$ =go to $g_{15,1} = 11$	PD
16	o	course materials	prepare for course	course materials	assistant
17	c		AND end	$s_{17} = 0$ $a_{17,1}$ =go to $g_{17,1} = 11$	assistant
18	o		course can be offered		Prof.

6 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.

This way, the process checklist also supports the key benefits of traditional WfMS. 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.

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

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 flexibility and parallelism. Here, process modellers have to choose a suitable transformation method as described in section 4. For future work we will evaluate the proposed approach within a real life business case. Here, we expect useful experiences regarding the acceptance and cooperation of participating agents. Based on these results we will improve methodology, design and representation. Furthermore, we will focus the transformation of loosely-specified processes like declarative process models, e.g., Declare [14].

Acknowledgement. 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 Heinrich Baumann is supported by Hanns-Seidel-Stiftung e.V.

References

1. Jablonski, S.: Do We Really Know How to Support Processes? Considerations and Reconstruction. In: Engels, G., Lewerentz, C., Schäfer, W., Schürr, A., Westfechtel, B. (eds.) Nagl Festschrift. LNCS, vol. 5765, pp. 393–410. Springer, Heidelberg (2010)

2. Zairi, M.: Business Process Management: a Boundaryless Approach to Modern Competitiveness. Business Process Management Journal 3, 64–80 (1997)
3. Reichert, M., Weber, B.: Enabling Flexibility in Process-aware Information Systems. Springer, Heidelberg (2012)
4. Van der Aalst, W., Weske, M., Grünbauer, D.: Case handling: A new paradigm for business process support. Data & Knowledge Engineering 53(2), 129–162 (2005)
5. Condon, C.: The Computer Won't Let Me: Cooperation, Conflict and the Ownership of Information. In: CSCW, pp. 171-185 (1993)
6. Luff, P., Heath, C., Greatbatch, D.: Tasks-in-interaction: paper and screen based documentation in collaborative activity. In: CSCW (1992)
7. Melenovsky, M.: Business process managements success hinges on business-led initiatives, Gartner Research, Stamford, CT, pp. 1-6 (July 2005)
8. Object Management Group Inc.: Business Process Model and Notation (BPMN) Version 2.0 (2011), `http://www.omg.org/spec/BPMN/2.0`
9. Wolff, A., Taylor, S., McCabe, J.: Using checklists and reminders in clinical pathways to improve hospital inpatient care. MJA 2004 181, 428–431 (2004)
10. Hales, B.M., Pronovost, P.J.: The checklist – A tool for error management and performance improvement. Journal of Critical Care 21(3), 213–235 (2006)
11. Hauser, R., Friess, M., Küster, J.M., Vanhatalo, J.: Combining Analysis of Unstructured Workflows with Transformation of Structured Workflows. In: Proc. 10th IEEE International Enterprise Distributed Object Computing Conference, EDOC (2006)
12. Koehler, J., Hauser, R., Küster, J., Ryndina, K., Vanhatalo, J., Wahler, M.: The Role of Visual Modeling and Model Transformation in Business-driven Development. In: Proc. 5th International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT 2006), Vienna, Austria (2006)
13. zur Muehlen, M., Recker, J.: How Much Language Is Enough? Theoretical and Practical Use of the Business Process Modeling Notation. In: Bellahsène, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 465–479. Springer, Heidelberg (2008)
14. Pešić, M.: Constraint-Based Workflow Management Systems, Shifting Control to Users (2006)