Behaviour and Execution Semantics of Extended Sequence Edges in Business Processes

Thomas Bauer
Hochschule Neu-Ulm, University of Applied Sciences, Wileystr. 1, 89231 Neu-Ulm, Germany

Keywords: Business Process, Flexibility, Control Flow, Sequence, Time, Workflow Engine.

Abstract: At business processes (BP), activities are usually considered as atomic units. This results in unnecessary restrictions, e.g. when modelling sequences of activities. Here, flexibility can be increased by allowing that a sequence edge refers to the start and to the end events of their source and target activity arbitrarily. This allows additional execution orders at the runtime of the BP, i.e. the end users have more flexibility at BP execution. Nevertheless, we respect all modelled control flow conditions, as well as time constraints defined between activities (e.g. minimum time intervals). A process engine requires a formal execution semantics, to be able to control such a BP automatically. Therefore, in this paper, we develop corresponding execution rules. Furthermore, we present measures that enable the process engine to delay and to speed up the start and the completion of activities in order to respect the modelled time constraints.

1 INTRODUCTION

The aim of our project CoPMoF (Controlable Pre-Modeled Flexibility) is to increase the flexibility at the execution of business processes (BP). However, dynamic changes (Reichert and Weber, 2012) shall not be used for this purpose. Instead, flexibility requirements, that are expectable for a BP, are pre-modelled already at build time. Later, at runtime, it is only necessary to use (apply) this flexibility. This is much easier for the end users and causes less effort. In addition, such flexibility is only available at process points where it is desired in fact (i.e. intended and approved by the responsible person). The CoPMoF project investigates which types of flexibility are usually required and how they can be pre-modelled. This concerns the control flow of BP (e.g. optional and alternative activities) (Bauer, 2024, 2021, 2020) as well as other process perspectives (e.g. alternative actor assignments) (Bauer, 2019).

A possibility to increase flexibility is to extend the capabilities of sequence edges: Normally, the activities of a BP are considered as atomic units at BP modelling (Russell and Hofstede, 2006). Therefore, at a sequence edge, the preceding Act. A must be completed before the succeeding Act. B can be started (cf. Fig. 1a). In more detail, however, an activity consists of a start event and an end event. This level of detail is typically used for logging (i.e. writing the log file) in process management systems (PMS). Furthermore, such events are used at process mining (Dakic et al., 2018; Zerbino et al., 2021). However, these events usually cannot be used arbitrarily to model control flow edges (Russell and Hofstede, 2006). Instead, sequence edges must always start from an end event of an activity and must have a start event as target (cf. Fig. 1a). This shall be extended so that they can start and end at any event type (Fig. 1b). With the depicted edges, the execution order ii) becomes possible, additionally. Furthermore, the approach presented in this paper allows to define arbitrary temporal dependencies between start and end events of activities.

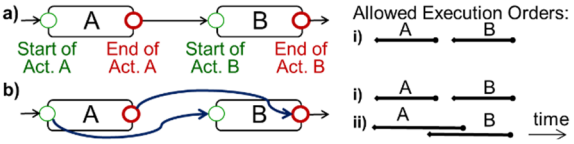

Figure 1: 1a) Classic Sequence Edge b) Sequence Edges with Extended Behaviour.

A BP for the development of electronic components (e.g. of a vehicle) is used to explain how the additional edge types can be used (cf. Fig. 2a). From Act. A to Act. B the classic sequence Edge ① was modelled (edge type EndBeforeStart with the meaning the end event of the preceding activity must occur before the succeeding activity can be started.). In this BP, after the design of the overall architecture

Bauer, T.
Behaviour and Execution Semantics of Extended Sequence Edges in Business Processes.
DOI: 10.5220/0013272200003929
In *Proceedings of the 27th International Conference on Enterprise Information Systems (ICEIS 2025) - Volume 2*, pages 829-836
ISBN: 978-989-758-749-8; ISSN: 2184-4992

(Act. B), the control units are developed (Act. C). To save development time (concurrent engineering), however, these activities shall be performed in an overlapping manner. For this reason, the type "StartBeforeStart" was used for the Edge ②. This means that the execution of Act. B must start before the start of Act. C. Therefore, Act. C can already be started after the start of Act. B, i.e. it does not have to wait for its completion. The same applies to Act. D in relation to Act. C (Edge ③). However, because of Edge ④, it is additionally defined that the test of the developed control units can only be completed after their development (Act. C) was completed. Finally, the Edge ⑤ specifies a maximum time for the execution of the whole BP, because Act. D must be completed at least 120 days after the start of Act. A.

The additional edge types can also be used in combination with more complex process structures as Split- and Join-Nodes. The (condensed) BP for vehicle delivery of Fig. 2b is used to explain the purpose of such edges. Vehicle production is completed with the final inspection (Act. A). Then, vehicle delivery and invoicing take place in parallel: A vehicle, that shall be delivered to a customer, is cleaned by the truck driver in Act. B (e.g. removing any rubbish left in the vehicle). This must be done before the transport (Act. C) is completed since he is not able to clean the vehicle afterwards. However, it is allowed that Act. B and Act. C are performed in an overlapping manner, e.g. by cleaning the vehicle during a transport break. Therefore, the type EndBeforeEnd is used for Edge ③. In addition, the vehicle transport (Act. C) must start before Act. D (inform customer about delivery) starts (StartBeforeStart for Edge ④). In the case of an earlier notification, the risk of a misinformation would be too high, because before the vehicle transport starts, there is still a high probability that the transport will be cancelled (e.g. because the truck is not available or is broken). Several edges with a new edge type have an AND-Join (end of parallelism) as target: Before the Act. F (handover to customer) can be started, Act. C and Act. D must be completed (⑤ and ⑥). In addition, it is necessary that Act. E was at least started (Edge ⑧), i.e. an employee of the branch was instructed to create the invoice. But, in order to prevent that the customer has to wait unnecessarily, then Act. F can be started already (e.g. with the explanation of vehicle functions). However, this Act. F can only be completed after the invoice has been handed over. Since it must be created and printed beforehand, the completion of Act. F requires the completion of Act. E (EndBeforeEnd at Edge ⑨).

Similar requirements are also known from project management (Wysocki, 2019). There, sequence dependencies can refer to any start and end events of tasks, as well. This results in four possible combinations: finish-to-start, finish-to-finish, start-to-start, and start-to-finish (Wysocki, 2019). These correspond to the four types of sequence edges of CoPMoF (cf. Section 2.1). In addition, minimum and maximum time intervals between start and end events can be defined at project management, as well. Their purpose is to determine the critical path of a project, i.e. the required execution time. In contrast, at CoPMoF, the new edge types reduce the execution time of a BP by enabling more parallelism at the execution of activities. Of course, defined time constraints shall also be respected. Similar orders, as enabled by the new edge types we introduce with CoPMoF, are also described in Allen's interval algebra (Allen, 1983) (e.g. overlaps and during, cf. Fig. 3c).

(Bauer, 2023) presents examples of BP from practice, where extended sequence edges are useful. Furthermore, the resulting requirements were explained. However, until now, there exists no research that develops a method that enables a PMS to control the execution of corresponding BP (at runtime). This paper handles this research gap: It describes how a PMS can delay or speed up the start or end of activities. In addition, we develop an execution semantics for BP with extended sequence edges and time conditions. The latter is required by the PMS to determine activities, that are executable concerning the control flow, and the associated points in time.

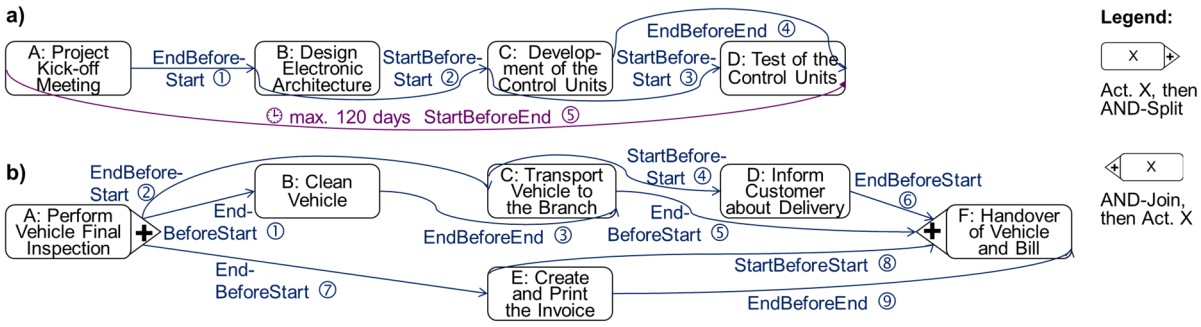

Figure 2: Example Processes with Extended Sequence Edges and a Time Edge.

In Section 2, relevant preliminary work from the project CoPMoF is presented and the general state of the art is discussed. Section 3 explains the behaviour of the PMS at runtime. For this purpose, in particular, a formal execution semantics is defined. The article concludes with a summary and an outlook.

2 BASICS AND LITERATURE

This section presents preliminary work of our project CoPMoF and the state of research and technology.

2.1 Preliminary Work

The basic idea of extended sequence edges is that they can arbitrarily refer to the start and end events of their source and target activities. We have already published this idea in (Bauer, 2023). Furthermore, this paper presents example scenarios where extended sequence edges are required. They are used to demonstrate that (the already mentioned) four types of sequence edges, temporal constraints with minimum and maximum durations, and the combination of these are required, in practice.

However, the meaning of the four edge types was defined only informally. Similar as in Fig. 3, for each type, the possible execution orders were listed and the intended behaviour was explained:

1. EndBeforeStart: The end of Act. A must happen before the start of Act. B. This corresponds to a classic sequence edge, cf. (Russell and Hofstede, 2006).

2. EndBeforeEnd: The end of Act. A must happen before the end of Act. B

3. StartBeforeStart: The start of Act. A must happen before the start of Act. B

4. StartBeforeEnd: The start of Act. A must happen before the end of Act. B

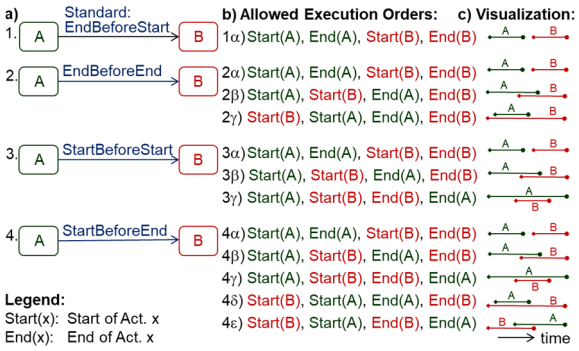

Figure 3: Types of Sequence Edges between Act. A and B.

The behaviour of the time constraints was described only in text form, as well, since the meaning of a minimum or maximum time interval is obvious.

The process engine of a PMS requires an algorithm to be able to control the execution of BP instances. An explanation, that can only be understood by humans, is not sufficient for this purpose. Therefore, in this paper, we develop a formal execution semantics (i.e. rules) for activity instances, that can be used by such a process engine. These rules consider the four types of sequence edges as well as minimum and maximum time intervals between arbitrary start and end events of activities.

2.2 Technology and Research

Commercial PMS are often based on standardized BP modelling languages such as BPEL (OASIS, 2007) and BPMN (OMG, 2011). These standards provide sequence edges that allow only pure sequential orders of activities (Type 1 in Section 2.1). An AND-Split enables overlapping execution, but each execution order is allowed for activities of different parallel branches. There are no building blocks that realize the behaviour of the Types 2 to 4 introduced in the previous section (e.g. StartBeforeStart).

In BPMN, maximum time intervals can be realized by an intermediate timer event (OMG, 2011), additional paths, and an escalation activity. This allows to enforce maximum time intervals. However, such an approach results in complex process graphs that may be too confusing for “normal BP designers”.

(Russell and Hofstede, 2006) present control flow patterns that enable many execution orders of activities. As mentioned above, activities are considered as atomic units. Therefore, sequence edges cannot refer to arbitrary start and end events of activities. Thus, the Types 2 to 4 of Section 2.1 are not respected.

At case handling (Aalst et al., 2005), the state of activity input data determines whether it can be started. An actor can decide to start an activity as soon as the required input data are available. (Hewelt and Weske, 2016) extend such approaches by allowing to model a lifecycle for data and to introduce own activity execution states. Both can be used to define when an activity can be started, i.e. this is not defined by control flow edges. With this approach, StartBeforeStart dependencies (Type 3) can be realized.

Constraint-based approaches (Reichert and Weber, 2012) define the control flow with rules that restrict the allowed execution orders. Since these constraints refer to whole activities, dependencies of Types 2 to 4 cannot be modelled here, as well.

(Heinlein, 2001) enables the definition of arbitrary dependencies between the start and the end of activities (including Types 2 to 4). However, this approach does not define dependencies between activities of the same process instance, but between activities of different process instances and even different process templates. As such dependencies do not belong to a single process graph, they cannot be modelled as (graphical) control flow edges. Instead, regular expressions are used for this purpose.

(Lanz et al., 2010) presents design patterns for defining minimum and maximum time intervals between activities. They can arbitrarily refer to the start and the end events of their predecessor and their successor activity. Thus, all four edge types of Section 2.1 are covered. However, the temporal constraints are not discussed in the context of control flow edges and no execution semantics is presented.

3 BEHAVIOR AT RUNTIME

This section explains how a PMS can control BP instances that contain the new types of edges. First, we describe how the start or completion of activities can be delayed or accelerated. Then, after defining some basics, a formal execution semantics is presented.

3.1 Activity Start and end Times

If an activity shall not be started yet (e.g. due to a minimum time interval in Case A2 of Fig. 4), it is not inserted into the worklists of the users (as always at PMS). If an activity can be started earlier (e.g. due to a new type of sequence edge in Case B1), a corresponding entry is created already. If the start of an activity must happen earlier (Case B2), the PMS "enforces" this through escalations (Aalst et al., 2007). Escalations were already used by many PMS today. Thereby, messages were sent (automatically and timely) to the potential actors, to their supervisors, or to process administrators, in order to remind them of the upcoming deadline. The timely completion of activities (D2) can be "enforced" through escalations, as well. If an activity must not be completed yet (C1 and C2), for example, the "end button" of the corresponding activity programme can be disabled ("greyed out"). As this may confuse the user, additional information should be displayed that explains why it is not yet allowed to complete this activity.

Fig. 4 contains cases where additional execution rules are required for the process engine (e.g. Case B1). These rules are explained in the following. To improve readability, the numbers of these new rules are shown in Fig. 4. For cases where an additional execution state for activity instances becomes necessary (A2 and C1/2), this is also depicted in Fig. 4:

because of	Start of Act. B		End of Act. B	
	A: later	B: earlier	C: later	D: earlier
1: Sequence Edge	case does not exist	StartBefore-Start A B Rule 3, 8	StartBeforeEnd A B EndBe-foreEnd A B Rule 5, 6, new State RunningCompletable	case does not exist
2: Time Constraint	min. x B Rule 1, 2, 8, new state WaitingForTime	max. x B Rule 4, enforced by an escalation	min. x B Rule 5', new State RunningCompletable	max. x B Rule 7, enforced by an escalation

Figure 4: Overview of the Cases.

3.2 Definitions

In the following, the usual definitions (OMG, 2011; Reichert and Dadam, 1998; Weske, 2019) for BP and execution states of activity instances are extended, so that they can be used as a basis for developing execution rules for the new edge types.

Def. 1: Process template and instance, activity states:
A process template PT=(N, C, T) consists of a set of nodes N, a set of control flow edges C, and a set of time edges T.
A process instance PI=(PT, State) consists of a process template PT and a set of activity states (State). Each activity a∈N can have a different state State(a). Activities can have the states shown in Fig. 6, i.e. ∀a∈N applies: State(a)∈ {Inactive, WaitingForTime, Active, Running, RunningCompletable, Completed}

The PMS knows the current state of each activity and changes these states during the execution of the process instance by using predefined execution rules. This allows the PMS to control the execution order of the activities, insert entries into worklists, start activity programs (e.g. forms), call automatically executed services, etc.

Def. 2: A control flow edge c∈C is defined as c = (SourceAct, TargetAct, Type) with:
SourceAct: the source activity of the edge
TargetAct: the target activity of the edge
Type: the type of edge with Type∈ {StartBeforeStart, StartBeforeEnd, EndBeforeStart, EndBeforeEnd}

In Def. 1, control flow and time edges are realized as different types of edges. The reason for this separation is that the existence of a time edge between two activities does not always imply that there exists a sequence edge with the same type: In Fig. 5, the start of Act. B must occur at latest (i.e. max.) 10 hours after the end of Act. A (time edge with the classic type EndBeforeStart). However, to speed up the BP execution, it is allowed that Act. B starts before the end of Act. A, i.e. there is no control flow edge of this type

EndBeforeStart between these activities (but a control flow edge with type StartBeforeStart, cf. Fig. 5).

Figure 5: Example for Activities Connected with a Time Edge but not with a Control Flow Edge of the same Type.

Def. 3: A time edge t∈ T is defined as t ={SourceAct, TargetAct, Type, MinTime, MaxTime} with:
SourceAct, TargetAct, and Type: as defined in Def. 2
MinTime: the minimum time interval defined for this edge
MaxTime: the maximum time interval defined for this edge
For edges without a corresponding time constraint, MinTime or MaxTime has the value undef.

To be able to use this information in the execution rules, it can be accessed by functions with the same name. For example, SourceAct(c) returns the source activity of the edge c and State(a) returns the current execution state of Act a.

To realize the new types of sequence edges, we extend the set of states of an activity instance. The two additionally required states are coloured in Fig. 6. Furthermore, edges are labelled with the numbers of the corresponding rules (cf. Sections 3.3 to 3.7) that are required to realize the new edge types. Rules that are already known from classical PMS (OMG, 2011; Reichert and Dadam, 1998; Weske, 2019), are not repeated in the following. Therefore, Fig. 6 also contains edges without labels.

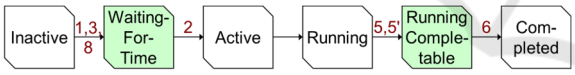

Figure 6: State Transition Diagram for Activity Instances.

3.3 Delayed Activity Start

Sequence Edge: It is not possible that an Act. B becomes startable later because of a new sequence edge type, than with classic edges: The only new type that refers to the start of its target Act. B is StartBeforeStart. With an edge of this type, e.g. from Act. A to Act. B (Case B1 in Fig. 4), Act. B can be started earlier than with the classic edge type EndBeforeStart.
Time Edge: An Act. B can become startable later because it is the target activity of a time edge with a predefined minimum time (A2 in Fig. 4). Since the delay concerns the start of this activity, the edge can have the type EndBeforeStart or StartBeforeStart.

To enable the PMS to handle such time edges at runtime, the additional activity state WaitingForTime is required (cf. Fig. 6): The process engine "remembers" the fact that the preceding activity of Act. B was already completed, by leaving the start state Inactive for Act. B. If only control flow edges were respected, the Act. B would be startable now, i.e. with the "classic" execution rules its state would change to Active. However, this is not yet allowed because of the modelled time constraint. This is signalled by the new state WaitingForTime. The classic execution rule is modified in such a way that the state Inactive is not directly followed by Active. Instead, the state changes to WaitingForTime first:
Rule 1: The edge c∈C is a "normal" control flow edge with the target activity a, i.e. TargetAct(c)=a ∧ Type(c)=EndBeforeStart.
After completion of the source activity s of this edge, Act. a changes to the state WaitingForTime:
If the Act. s=SourceAct(c) reaches State(s)=Completed, then the state of Act. a changes to State(a)= WaitingForTime.

The transition from this state WaitingForTime to Active is performed by Rule 2, as soon as all the minimum waiting times of the incoming time edges have been reached. Then, Act. a can be actually executed.
Rule 2: Let T_a be the set of time edges that are relevant for the calculation of the earliest start time of an Act. a with State(a)=WaitingForTime:
$T_a = \{t_i \in T \mid TargetAct(t_i)=a \wedge Type(t_i) \in \{StartBeforeStart, EndBeforeStart\} \wedge MinTime(t_i) \neq undef\}$
Then, the earliest start time of Act. a results as the largest (i.e. latest) time, that results from one of these edges: $EarliestStartTime_a = Max(ResultingTime_{Min,Start}(t_i))\ \forall t_i \in T_a$
For this purpose, the time $ResultingTime_{Min,Start}(t)$ that results for the edge t is calculated by adding the minimum time defined for this edge t to the start or the end time (depending on the edge type) of the source activity of edge t:

$$ResultingTime_{Min,Start}(t_i) = \begin{cases} StartTime(SourceAct(t_i)) + MinTime(t_i) \\ \qquad \text{if } Type(t_i) = StartBeforeStart \\ EndTime(SourceAct(t_i)) + MinTime(t_i) \\ \qquad \text{if } Type(t_i) = EndBeforeStart \end{cases}$$

As soon as the $EarliestStartTime_a$ is reached (i.e. $CurrentTime \geq EarliestStartTime_a$), the state of Act. a is changed to Active: State(a) = Active.

Remark: If there do not exist such time edges (i.e. $T_a=\{\}$), the calculation of the maximum (Max) results in $EarliestStartTime_a=-\infty$. Since $CurrentTime \geq -\infty$ always holds, Act. a changes directly to State(a)=Active, as soon as State(a)=WaitingForTime is reached. This means that the state WaitingForTime is then not relevant since it is left immediately.

The following variants of ResultingTime are required by Rules 4, 5' and 7: $ResultingTime_{Max,Start}(t_i)$ is calculated as described in Rule 2, but MaxTime(t_i)

(instead of MinTime(t_i)) is added. Furthermore, ResultingTime$_{Min,End}$(t_i) and ResultingTime$_{Max,End}$(t_i) are calculated in the same way, but T_a contains the edges t_i with Type(t_i)∈ {StartBeforeEnd, EndBeforeEnd}.

3.4 Earlier Activity Start

Sequence Edge: An Act. B can become executable because of an edge of the type StartBeforeStart (Case B1 in Fig. 4). Similar as at a classic sequence edge (i.e. type EndBeforeStart, cf. Rule 1), its state must change from Inactive to WaitingForTime (not yet Active as explained in Section 3.3). This is realized by the following execution rule:

Rule 3: The Act. a with State(a)=Inactive is the target of the control flow edge c∈ C of type StartBeforeStart, i.e. TargetAct(c)=a ∧ Type(c)=StartBeforeStart.

After the start of the source activity s of this edge, the Act. a changes to the state WaitingForTime:

If the state of the activity s=SourceAct(c) changes to State(s)=Running, then the state of Act. a is set to: State(a)=WaitingForTime

Time Edge: If an Act. B has an incoming time edge with a maximum time (B2 in Fig. 4), its start must be enforced by escalations up to a certain point in time (LatestStartTime$_a$, see below). Thereby, only such time edges are relevant that refer to the start of this Act. B (i.e. types ...BeforeStart). The time LatestStartTime$_a$ is calculated as follows:

Rule 4: Let T_a be the set of time edges relevant for the calculation of the latest start time of Act. a:

T_a = {t_i∈ T | TargetAct(t_i)=a ∧ Type(t_i)∈ {StartBeforeStart, EndBeforeStart} ∧ MaxTime(t_i)≠undef}

Then, the latest start time of Act. a results as the smallest (i.e. earliest) time, that results from one of these edges: LatestStartTime$_a$ = Min(ResultingTime$_{Max,Start}$(t_i)) ∀t_i ∈ T_a

As mentioned, this latest start time shall be ensured by escalations. Since some time may elapse after an escalation is triggered, before it is recognized by the user and the activity is started in fact, the escalation should be triggered timely before LatestStartTime$_a$ is reached. In some scenarios, in addition, multi-level escalations (Aalst et al., 2007) can be useful: For example, first an email is sent to the potential actors of the activity. If the activity was not started after a certain time, a responsible person (supervisor, BP administrator) is informed. This must also be done timely before LatestStartTime$_a$ is reached.

3.5 Delayed Activity Completion

Sequence Edge: The sequence edges shown in case C1 of Fig. 4 (types ...BeforeEnd) can result in a delayed completion of their target B. That means, the user may have already executed Act. B, but is not allowed to finish it yet, because of such an edge. To enable the process engine to recognize this case, we introduce the new state RunningCompletable. During its execution, Act. B initially has the state Running. It can only be completed if its preceding Act. A, that is connected with such an edge, has been started or completed. Then, the state of Act. B is changed to RunningCompletable. Rule 5 realizes this state transition:

Rule 5: Let C_a be the set of control flow edges that are relevant for the completion of an Act. a with State(a)=Running:

C_a = {c_i∈ C | TargetAct(c_i)=a ∧ Type(c_i)∈ {StartBeforeEnd, EndBeforeEnd}}

A state change for a is allowed as soon as all source activities of these edges have been started (i) or finished (ii), i.e. all following conditions are fulfilled:

i) If ∀c_i∈ C_a with Type(c_i)=StartBeforeEnd applies: State(SourceAct(c_i)) ∈ {Running, RunningCompletable, Completed} and

ii) if ∀c_i∈ C_a with Type(c_i)=EndBeforeEnd applies: State(SourceAct(c_i)) = Completed,

then the state of Act. a is changed to:

State(a)=RunningCompletable

Remark: If there do not exist any control flow edges of these types, whose target activity is Act. a (i.e. C_a={}), then the conditions i) and ii) are fulfilled. Therefore, the state of Act. a changes to RunningCompletable immediately after Running is reached.

The following Rule 6 replaces the classic rule that handles the completion of an activity. The main difference is that Rule 6 uses the (new) state RunningCompletable instead of the state Running.

Rule 6: If an Act. a has the state State(a)=RunningCompletable, the actor can complete this activity. This results in a state change to: State(a)=Completed

If it is currently not allowed to complete an activity (i.e. it is still in the state Running instead of RunningCompletable), this can be signalled to the actor, for example, by deactivating the Ok / Complete button of the activity program or the corresponding entry in the menu. Additionally, a help text (defined at build time) shall be displayed, that explains why this activity cannot be finished yet. In order to enable such a behaviour of the activity program, the interface (API) of the PMS must provide functions to retrieve the mentioned help text and to determine whether a particular activity can be terminated yet. The function for the completion of an Act. a shall return an error if it is already called at State(a)=Running.

Time Edge: Time edges with a predefined minimum time also can delay the completion of the Act. B (cf. C2 in Fig. 4). Here, these edges refer to the end of

Act. B, i.e. they have a type ...BeforeEnd. To respect such edges, Rule 5 is extended by the additional condition (iii), which must be fulfilled as well:

Rule 5': Let T_a be the set of time edges with the target activity a (with State(a)=Running) that are relevant for the calculation of its earliest completion time:

$T_a = \{t_i \in T \mid TargetAct(t_i)=a \wedge Type(t_i) \in \{StartBeforeEnd, EndBeforeEnd\} \wedge MinTime(t_i) \neq undef\}$

Then, the earliest completion time of Act. a results as:

$EarliestCompletionTime_a = Max(ResultingTime_{Min,End}(t_i)) \; \forall \; t_i \in T_a$

iii) A state change for Act. a is allowed as soon as the following condition is fulfilled:

$CurrentTime \geq EarliestCompletionTime_a$

All conditions must be fulfilled to enable a state change of Act. a, i.e. i) and ii) of Rule 5 as well as iii) of Rule 5'. However, if no such time edges exist (i.e. $T_a=\{\}$), the calculation of the maximum results in the value -∞, again. Therefore, the ≥ condition for CurrentTime is fulfilled, and thus also the condition iii).

The same Act. a can be the source and the target activity of a time edge $t_i \in T_a$ with type StartBeforeEnd. This represents the important special case that a minimum execution duration is defined for this Act. a, e.g. the duration for an adhesive to dry. Rule 5' respects this special case as well.

3.6 Earlier Activity Completion

Sequence Edge: It is not possible that an Act. a can be completed earlier because of the new types of control flow edges, than at classic process models: At the latter, the Act. a can be completed at any time after its start. An earlier completion (i.e. before starting) does not make sense. Therefore, the new types of control flow edges cannot cause this case.

Time Edge: Time edges with a maximum time can demand that an Act. B must be completed before a certain point in time (cf. D2 in Fig. 4). These must be edges that refer to the completion of Act. B (i.e. types ...BeforeEnd). Again, the PMS ensures timely completion through escalations. Rule 7 calculates the latest possible point in time:

Rule 7: Let T_a be the set of time edges that are relevant for the calculation of the latest completion time of Act. a:

$T_a = \{t_i \in T \mid TargetAct(t_i)=a \wedge Type(t_i) \in \{StartBeforeEnd, EndBeforeEnd\} \wedge MaxTime(t_i) \neq undef\}$

Then the latest completion time $LatestCompletionTime_a$ of Act. a results as the smallest (i.e. earliest) maximum completion time, that results from one of these edges t_i:

$LatestCompletionTime_a = Min(ResultingTime_{Max,End}(t_i)) \; \forall \; t_i \in T_a$

3.7 Gateways

This section deals with process behaviour at gateways, since the rules presented so far do not fully cover this aspect. For activities that are located after a Split-Node (e.g. Act. B, C, and E in Fig. 2b), no extensions are necessary, because such nodes have only one preceding activity (e.g. the Split-Activity A in Fig. 2b). Furthermore, the presented rules already cover the cases of multiple incoming edges of the types StartBeforeEnd and EndBeforeEnd (both by Rule 5) as well as multiple incoming time edges (Rules 2, 4, 5', and 7).

To improve readability, the case of multiple incoming sequence edges of the types EndBeforeStart and StartBeforeStart was not respected by the Rules 1 and 3. This case occurs, for example, at an AND-Join[1] (cf. Act. F in Fig. 2b)[2]. Such an activity must wait for several preceding activities. To enable this behaviour, the Rules 1 and 3 are replaced by Rule 8 (shown below). Here, the set S_{End} contains all preceding activities that are connected with edges of the type EndBeforeStart and S_{Start} activities connected with type StartBeforeStart. Rule 8 respects that not only one preceding Act. s (cf. Rules 1 and 3) must reach the required state, but all activities of these two sets. In the example of Fig. 2b, for Act. F the sets result as $S_{End}=\{C, D\}$ and $S_{Start}=\{E\}$ (the latter because of Edge ⑧; the Edge ⑨ is irrelevant for the startability of Act. F).

Rule 8: Let C_{End} be the set of "normal" control flow edges with Act. a as target:

$C_{End} = \{c_i \in C \mid TargetAct(c_i)=a \wedge Type(c_i)=EndBeforeStart\}$

[1] OR- and XOR-Join-Nodes have multiple incoming edges, as well. For these, only those predecessor activities of the Join-Node are relevant (i.e. are respected) that are located in paths that are actually executed at this process instance. Therefore, the sets S_{End} and S_{Start} shall not contain activities from paths that are not executed.

[2] In contrast to BPMN, Fig. 2b does not contain separate gateway nodes (e.g. as in ADEPT (Reichert and Dadam, 1998)), because the execution of a gateway does not require significant time. That means, its start and end events occur almost simultaneously. Therefore, in our context, the additional gateway node and the additional edge would be irrelevant. (In Fig. 2b, using gateway nodes would result in an AND-Join-Node with the incoming Edges ⑤,⑥, ⑧ and an additional outgoing edge to Act. F. The type (End- or StartBeforeStart) of this edge is irrelevant, because the start and the end events of the gateway occur almost at the same time.)

Let C_{Start} be the set of edges of type StartBeforeStart and Act. a as target:
$C_{Start} = \{c_i \in C \mid TargetAct(c_i)=a \wedge Type(c_i)=StartBeforeStart\}$
After the completion of all source activities $s \in S_{End}$ connected to Act. a with edges of C_{End} and after the start (i.e. state is at least Running) of all activities $s \in S_{Start}$ connected with edges of C_{Start}, Act. a changes to the state WaitingForTime:
If $\forall s \in S_{End}$ with $S_{End} = \{s \in N \mid \exists c_i \in C_{End} \wedge s = SourceAct(c_i)\}$ holds State(s)=Completed and
if $\forall s \in S_{Start}$ mit $S_{Start} = \{s \in N \mid \exists c_i \in C_{Start} \wedge s = SourceAct(c_i)\}$ holds State(s) $\in$ {Running, RunningCompletable, Completed},
then the state of Act. a changes to:
State(a)=WaitingForTime

4 SUMMARY AND OUTLOOK

The presented approach extends sequence edges by allowing that they use the start and the end events of their source and target activities arbitrarily. Furthermore, minimum and maximum time intervals can be defined, which can also refer to these events arbitrarily. We explain how a PMS can influence users (e.g. through escalations) in such a way that all these modelled conditions are met. In addition, the formal execution semantics of process engines is extended by introducing additionally required activity instance states and by defining further execution rules. This enables a PMS to automatically control BP that contain edges of the new types.

The presented rules still have to be evaluated technically by a prototype implementation. For this purpose, ideally, they will be integrated into an existing PMS that can be used in practice. This would also allow an evaluation of their suitability for BP designers and end users. However, due to the complexity of process engines, such an integration can usually only be realized by the vendor of the PMS. This is the long-term goal, as it makes the described functionalities available to many users. An integration into a BP modelling tool for pure BP documentation and optimization (e.g. as an extension of BPMN (Bauer, 2025)) would be less complex. Even this is useful because it enables BP modelling with more details (i.e. advanced activity orders and time intervals). By analysing the resulting BP models, later on, it can be determined how often the new edge types are required in practice. High demand may motivate PMS vendors to implement them in their process engines.

REFERENCES

Aalst, W.M.P. van der, Rosemann, M., Dumas, M., 2007. Deadline-based Escalation in Process-Aware Information Systems. Decision Support Systems 492–511.

Aalst, W.M.P. van der, Weske, M., Grünbauer, D., 2005. Case Handling: A New Paradigm for Business Process Support. Data & Knowledge Engineering 53, 129–162.

Allen, J.F., 1983. Maintaining Knowledge about Temporal Intervals. Communications of the ACM 26, 832–843.

Bauer, T., 2025. Extending BPMN to Enable the Pre-Modelling of Flexibility for the Control Flow of Business Processes, Proc. ICEIS25. Porto

Bauer, T., 2024. A Formal Execution Semantics for Sophisticated Dynamic Jumps within Business Processes, Proc. ICEIS24. Angers, 634–642.

Bauer, T., 2023. Modelling of Advanced Dependencies Between the Start and the End of Activities in Business Processes. Proc. ICEIS23. Prague, 457–465.

Bauer, T., 2022. Requirements for Dynamic Jumps at the Execution of Business Processes. Proc. 12th Int. Symposium on Business Modeling and Software Design.

Bauer, T., 2021. Pre-modelled Flexibility for the Control-Flow of Business Processes, in: Enterprise Information Systems. Springer, pp. 833–857.

Bauer, T., 2020. Business Processes with Pre-designed Flexibility for the Control-Flow. Proc. ICEIS20, 631–642.

Bauer, T., 2019. Pre-modelled Flexibility for Business Processes. Proc. ICEIS19. Heraklion, 547–555.

Dakic, D., Stefanovic, D., Cosic, I., Lolic, T., Medojevic, M., 2018. Business Process Mining Application: A Literature Review. Proc. 29th DAAAM Int. Symposium on Intelligent Manufacturing and Automation 866–875.

Heinlein, C., 2001. Workflow and Process Synchronization with Interaction Expressions and Graphs. Proc. 17th Int. Conf. on Data Engineering 243–252.

Hewelt, M., Weske, M., 2016. A Hybrid Approach for Flexible Case Modeling and Execution. Proc. 14th Int. Conf. on Business Process Management 38–54.

Lanz, A., Weber, B., Reichert, M., 2010. Workflow Time Patterns for Process-Aware Information Systems. Proc. Enterprise, Business-Process, and Information 94–107.

OASIS, 2007. Web Services Business Process Execution Language Version 2.0.

OMG, 2011. Business Process Model and Notation (BPMN) 2.0.

Reichert, M., Dadam, P., 1998. ADEPTflex - Supporting Dynamic Changes of Workflows Without Losing Control. Journal of Intelligent Information Systems 10.

Reichert, M., Weber, B., 2012. Enabling Flexibility in Process-Aware Information Systems. Springer.

Russell, N., Hofstede, A.H.M., 2006. Workflow Control-Flow Patterns: A Revised View. BPM Center Report.

Weske, M., 2019. Business Process Management: Concepts, Languages, Architectures, 3rd ed. Springer.

Wysocki, R.K., 2019. Effective Project Management. Wiley.

Zerbino, P., Stefanini, A., Aloini, D., 2021. Process Science in Action. Technological Forecasting and Social Change 172.