M. Riebisch, M. Tropmann-Frick (Hrsg.): Modellierung 2022,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2022 155

Verhalten und Ausführungssemantik von erweiterten Sequenzkanten in Geschäftsprozessen

Thomas Bauer[1]

Abstract: Bei der Modellierung von Geschäftsprozessen (GP) werden Aktivitäten üblicherweise als atomare Einheiten betrachtet. Dies führt zu unnötigen Einschränkungen, wenn z.B. Sequenzen von Aktivitäten modelliert werden. Hier kann die Flexibilität erhöht werden, indem sich Sequenzkanten beliebig auf die Start- und Endeereignisse ihrer Quell- und Zielaktivitäten beziehen können. Dadurch werden zur Ausführungszeit der GP (Runtime) zusätzliche Ausführungsreihenfolgen möglich, d.h. die Benutzer haben mehr Flexibilität bei der Prozessbearbeitung. Dennoch werden selbstverständlich alle modellierten Kontrollflussbedingungen ebenso eingehalten, wie zeitliche Constraints zwischen Aktivitäten (z.B. Mindestzeitabstände). Damit eine Prozess-Engine einen GP entsprechend dieser Bedingungen (automatisch) steuern kann, ist eine formale Ausführungssemantik erforderlich. Sie wird in diesem Beitrag ebenso vorgestellt, wie Maßnahmen, mit denen die Prozess-Engine den Start bzw. die Beendigung von Aktivitäten verzögern und beschleunigen kann, um so das gewünschte Verhalten bei der GP-Ausführung zu erreichen.

Keywords: Geschäftsprozess; Flexibilität; Kontrollfluss; Sequenz; Zeit; Workflow-Engine

1 Einleitung

Ziel des Projektes CoPMoF (Controlable Pre-Modeled Flexibility) ist, die Flexibilität von PAIS (Process-aware Information System) zu erhöhen. Allerdings werden hierzu keine dynamischen Änderungen (vgl. [RW12]) eingesetzt. Stattdessen wird in einem Geschäftsprozess (GP) zu erwartender Flexibilitätsbedarf bereits zur Buildtime vor-modelliert. Zur Runtime muss diese Flexibilität dann nur noch genutzt (angewandt) werden, was für den Endbenutzer deutlich einfacher und mit weniger Aufwand durchzuführen ist. Außerdem ist Flexibilität dadurch nur an tatsächlich erwünschten (d.h. vom Prozessverantwortlichen vorgesehenen) Stellen verfügbar und es können Benutzerrechte für deren Anwendung vergeben werden. Im Projekt CoPMoF wurde untersucht, welche Arten von Flexibilität üblicherweise benötigt und vormodelliert werden können. Dabei wurden sowohl Flexibilitätsbedarfe für den Kontrollfluss von GP (z.B. optionale und alternative Aktivitäten) [Ba21b] als auch für andere Prozessperspektiven (z.B. alternative Bearbeiterzuordnungen) [Ba19b] betrachtet.

Eine der Möglichkeiten, um die Flexibilität des Kontrollflusses zu erhöhen, ist es, die Mächtigkeit von Sequenzkanten zu erweitern: Normalerweise werden Aktivitäten von GP bei der Modellierung des Kontrollflusses als atomare Einheiten betrachtet [RH06]. Bei einer Sequenz muss deshalb die Vorgängeraktivität A abgeschlossen sein, bevor

[1] Hochschule Neu-Ulm, Fakultät Informationsmanagement, Wileystr. 1, 89231 Neu-Ulm, thomas.bauer@hnu.de

doi:10.18420/modellierung2022-003

die Nachfolgeraktivität B gestartet werden kann (vgl. Abb. 1a). Detaillierter betrachtet besteht eine Aktivität jedoch aus einem Start- und einem Endeereignis. Auf diesem Detaillierungsgrad erfolgt bei Prozess-Management-Systemen (PMS) üblicherweise die Protokollierung (Log-File) und solche Ereignisse werden zum Process-Mining verwendet [Da18, ZSA21]. Allerdings können diese Ereignisse normalerweise nicht beliebig zur Modellierung von Kontrollflusskanten verwendet werden [RH06]. Stattdessen müssen Sequenzkanten immer an einem Endeereignis starten und ein Startereignis als Ziel haben (vgl. Abb. 1a). Dies soll so erweitert werden, dass sie an beliebigen Ereignissen beginnen und enden können (Abb. 1b). Im dargestellten Beispiel ermöglicht dies zusätzlich die Ausführungsreihenfolge ii) aus Abb. 1c. Außer den Reihenfolgebeziehungen können bei dem in diesem Beitrag vorgestellten Ansatz auch noch zeitliche Abhängigkeiten beliebig zwischen den Start- und Endeereignissen definiert werden.

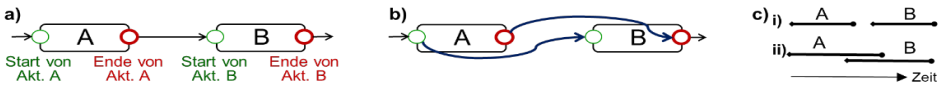

Abb. 1: a) Klassische Sequenzkante b) Sequenzkanten mit erweiterter Semantik c) dadurch erlaubte Ausführungsreihenfolgen

Der in Abb. 2a dargestellte GP zur Entwicklung von Elektronikkomponenten (z.B. eines Fahrzeugs) zeigt, wie die zusätzlichen Kantentypen verwendet werden können. Von der Akt. A zur Akt. B verläuft die klassische Sequenzkante ①. In dem GP findet nach dem Entwurf der Gesamtarchitektur (Akt. B) die Entwicklung der Steuergeräte (Akt. C) statt. Um Entwicklungszeit zu sparen (Concurrent Engineering), sollen diese Aktivitäten jedoch zeitlich überlappend ausgeführt werden. Deshalb wurde für die Kante ② der Typ „StartBeforeStart“ verwendet. Das bedeutet, dass die Bearbeitung der Akt. B vor dem Start von Akt. C beginnen muss, d.h. die Akt. C darf bereits ab dem Start von B gestartet werden, sie muss also nicht auf die Beendigung von Akt. B warten. Dasselbe gilt für die Akt. D in Bezug auf Akt. C (Kante ③). Durch die Kante ④ wird jedoch zusätzlich festgelegt, dass der Test der entwickelten Steuergeräte erst dann abgeschlossen werden darf, wenn zuvor deren Entwicklung (Akt. C) beendet wurde. Schließlich legt die Kante ⑤ eine Maximalzeit für die Ausführung des gesamten GP fest, weil zwischen dem Start der Akt. A und dem Ende der Akt. D maximal 120 Tage vergehen dürfen.

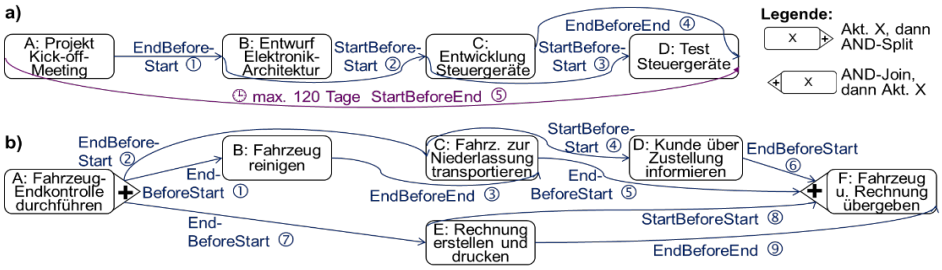

Abb. 2: Beispielprozesse mit erweiterten Sequenz- und Zeitkanten

Die zusätzlichen Kantentypen können auch in Kombination mit komplexeren Ablaufstrukturen verwendet werden. Abb. 2b zeigt einen GP zur Fahrzeugauslieferung, der stark verdichtet ist, um insb. den Zweck solcher Kanten zu erläutern. Die Fahrzeug-Herstellung wird mit der Endkontrolle (Akt. A) abgeschlossen. Danach findet parallel die Fahrzeug-Auslieferung und die Rechnungserstellung statt. Ein auszulieferndes Fahrzeug wird in Akt. B vom Fahrer endgereinigt (z.B. im Fahrzeug verbliebener Müll entfernen). Dies muss abgeschlossen sein, bevor der Transport (Akt. C) abgeschlossen wird, weil es nicht möglich ist, das Fahrzeug danach noch zu reinigen. Es ist jedoch erlaubt, dass die Aktivitäten B und C überlappend ausgeführt werden, z.B. indem das Fahrzeug während einer Transportpause gereinigt wird. Deshalb wird für die Kante ③ der Typ EndBeforeEnd verwendet. Außerdem muss der Fahrzeugtransport (Akt. C) beginnen, bevor die Akt. D (Kunde über anstehende Zustellung informieren) beginnt (StartBeforeStart bei Kante ④). Im Falle eines früheren Informierens wäre die Gefahr einer Fehlinformation zu groß, weil vor Beginn des Fahrzeugtransports die Wahrscheinlichkeit noch hoch ist, dass der Transport doch nicht stattfindet (z.B. weil der LKW nicht verfügbar oder defekt ist). ⑧ ist ein Beispiel für einen neuartigen Kantentyp, der zudem in einen AND-Join (Ende der Parallelität) mündet: Bevor die Akt. F (Übergabe an Kunde) gestartet werden darf, müssen die Aktivitäten C und D beendet sein. Außerdem muss die Akt. E zumindest gestartet worden sein, d.h. ein Mitarbeiter wurde mit der Rechnungserstellung beauftragt. Damit der Kunde nicht unnötig warten muss, darf mit Akt. F aber bereits begonnen werden (z.B. Erläuterung der Fahrzeugfunktionen). Abgeschlossen werden kann diese Akt. F aber erst nach der Übergabe der Rechnung. Da diese davor fertig erstellt und gedruckt sein muss, erfordert die Beendigung der Akt. F die Beendigung von Akt. E (EndBeforeEnd bei Kante ⑨).

Ähnliche Anforderungen sind auch aus dem Projektmanagement bekannt [Bu18]. So können sich auch dort Reihenfolgeabhängigkeiten auf beliebige Start- und Endeereignisse von Aufgaben beziehen. Dadurch ergeben sich vier Kombinationsmöglichkeiten: Normalfolge, Anfangsfolge, Endfolge, Sprungfolge [Bu18]. Diese entsprechen den vier Typen von Sequenzkanten in CoPMoF (vgl. Abschnitt 2). Außerdem können auch beim Projektmanagement minimale und maximale Zeitabstände zwischen den Start- und Endeereignissen definiert werden. Diese haben dort den Zweck, den kritischen Pfad eines Projektes, also die benötige Ausführungsdauer, zu ermitteln. Dahingegen haben die zusätzlichen Kantentypen (erweiterte Sequenzkanten) bei CoPMoF den Zweck, die Ausführungsdauer von GP zu reduzieren, indem mehr Parallelität zwischen den Aktivitäten ermöglicht wird. Hierbei sollen natürlich auch die zeitlichen Constraints eingehalten werden. Ähnliche zeitliche Reihenfolgen, wie sie durch die bei CoPMoF neu eingeführten Kantentypen ermöglicht werden, sind zudem in Allens Intervall Algebra [Al83] beschrieben (z.B. overlaps, during).

In [Ba19a] wurden Beispiele für GP vorgestellt, in denen erweiterte Sequenzkanten nützlich sind. Außerdem wurden die daraus resultierenden Anforderungen dargestellt. Allerdings wurde bisher nicht betrachtet, wie es einem PMS möglich ist, die Ausführung entsprechender GP zu steuern (zur Runtime). Diese Forschungslücke wird durch den vorliegenden Beitrag geschlossen: Er beschreibt, wie ein PMS den Start bzw. das Ende von Aktivitäten verzögern

bzw. beschleunigen kann. Außerdem wird eine Ausführungssemantik für GP mit erweiterten Sequenzkanten und Zeitbedingungen entwickelt. Diese wird von PMS benötigt, um die ausführbaren Aktivitäten und die zugehörigen Zeitpunkte zu ermitteln.

In Abschnitt 2 werden relevante Vorarbeiten aus dem Projekt CoPMoF vorgestellt und der allgemeine Stand der Literatur betrachtet. Abschnitt 3 beschreibt, wie sich das PMS zu Runtime verhält, insb. wird eine formale Ausführungssemantik definiert. Der Beitrag schließt mit einer Zusammenfassung und einem Ausblick.

2 Grundlagen und Literatur

Vorarbeiten: Die Grundidee erweiterter Sequenzkanten ist, dass sich diese beliebig auf die Start- und Endeereignisse ihrer Quell- und Zielaktivität beziehen können. Diese Idee wurde bereits in [Ba19a] publiziert. Dort wurden außerdem Beispielszenarien vorgestellt, in denen erweiterte Sequenzkanten benötigt werden. Aus diesen wurden (ähnlich wie das mit den in Abb. 2 dargestellten GP möglich wäre) die Anforderungen abgeleitet, dass vier Typen von Sequenzkanten, zeitliche Constraints mit minimalen und maximalen Zeitdauern und die Kombination daraus erforderlich sind.

Allerdings wurde in [Ba19a] die Bedeutung der vier Kantentypen lediglich informell festgelegt. So wie in Abb. 3 dargestellt, wurden für jeden Typ die erlaubten Ausführungsreihenfolgen aufgezählt und die Funktionsweise wurde erläutert:

1. EndBeforeStart	Ende von Akt. A muss vor dem Start von Akt. B erfolgen (das entspricht einer klassischen Sequenzkante, vgl. [RH06])
2. EndBeforeEnd	Ende von Akt. A muss vor dem Ende von Akt. B erfolgen
3. StartBeforeStart	Start von Akt. A muss vor dem Start von Akt. B erfolgen
4. StartBeforeEnd	Start von Akt. A muss vor dem Ende von Akt. B erfolgen

Ebenso wurde die Bedeutung der zeitlichen Constraints nur textuell beschrieben, da die Bedeutung eines minimalen bzw. maximalen Zeitabstandes intuitiv klar ist.

Die Prozess-Engine eines PMS benötigt einen Algorithmus zur Steuerung der auszuführenden GP-Instanzen. Eine nur für Menschen verständliche Erläuterung genügt hierfür nicht. Deshalb werden in dem vorliegenden Beitrag maschinell auswertbare Regeln entwickelt (d.h. formale Regeln), die eine Ausführungssemantik für Aktivitäteninstanzen definieren. Diese berücksichtigen sowohl alle vier Typen von Sequenzkanten als auch zeitliche Minimal- und Maximalabstände zwischen beliebigen Start- und Endeereignissen der Aktivitäten.

Stand von Forschung und Technik: Kommerzielle PMS basieren häufig auf standardisierten GP-Modellierungssprachen wie BPEL [OA07] und BPMN [Ob11]. Diese Standards definieren Sequenzkanten, die jedoch nur eine rein sequentielle Bearbeitung der Aktivitäten ermöglichen (Typ 1). Parallelitäten ermöglichen eine zeitlich überlappende Ausführung, jedoch ist dann jede beliebige Reihenfolge für Aktivitäten aus unterschiedlichen parallelen Zweigen erlaubt. Es gibt keine Konstrukte, die eine wie im vorherigen Abschnitt für die Typen 2 bis 4 vorgestellte Bedeutung haben (z.B. StartBeforeStart).

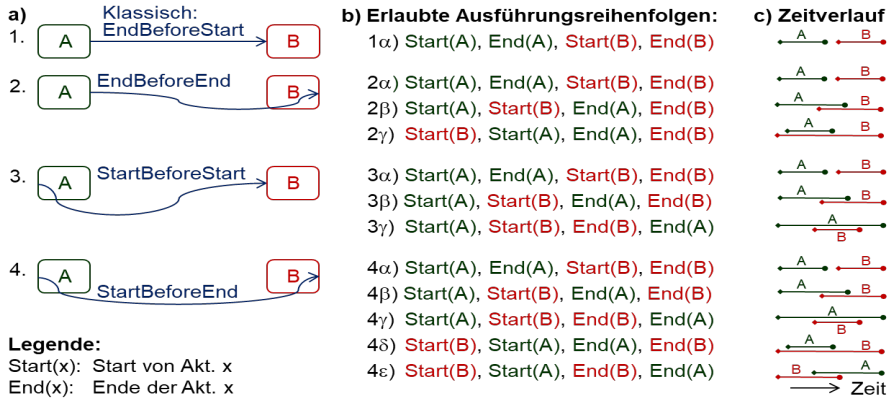

Abb. 3: a) Typen von Sequenzkanten zwischen den Aktivitäten A und B b) erlaubte Reihenfolgen von Start-/Endeereignissen der Aktivitäten c) graphische Darstellung dieser Reihenfolgen, vgl. [Al83]

In BPMN können Maximalzeiten mit einem intermediate Timer-Event [Ob11] realisiert werden. Hierzu müssen zusätzliche Ausführungspfade modelliert werden, die bei einer Zeitüberschreitung ausgeführt werden. Nach dem Timer-Event wird eine speziell für diesen Zweck vorgesehene Eskalationsaktivität modelliert, die beim Eintreten des Events (d.h. bei einer Zeitüberschreitung) automatisch ausgeführt wird. Dadurch können maximale Zeitabstände erzwungen werden. Durch eine solche Vorgehensweise ergeben sich jedoch komplexe Abläufe, mit deren Erstellung „normale GP-Modellierer“ evtl. überfordert sind.

Die in [RH06] vorgestellten Kontrollfluss-Pattern ermöglichen zahlreiche Ausführungsreihenfolgen von Aktivitäten. Die Aktivitäten werden hierbei als atomare Einheiten betrachtet, weswegen sich Sequenzkanten nicht auf beliebige Start- und Endeereignisse der verbundenen Aktivitäten beziehen können. Die Typen 2 bis 4 werden also nicht berücksichtigt.

Bei Ansätzen zum Case Handling [AWG05] definiert die Zustand der Eingabedaten einer Aktivität, ob diese gestartet werden kann. Dadurch kann ein Bearbeiter selbst entscheiden, eine Aktivität zu starten, sobald die erforderlichen Daten vorhanden sind. [HW16] erweitert solche Ansätze um die Möglichkeit, einen Lifecycle für Daten zu modellieren, hierbei eigene Ausführungszustände einzuführen und mittels dieser zu definieren, welche Aktivitäten startbar sind. Dies wird also nicht durch Kontrollflusskanten definiert. Dadurch kann man Abhängigkeiten vom Typ StartBeforeStart (Typ 3) realisieren: Für die Akt. F aus Abb. 2b kann z.B. modelliert werden, dass sie startbar ist, sobald die Rechnung den (benutzerdefinierten) Zustand „angelegt“ erreicht hat. Wenn eine Rechnung unmittelbar nach dem Start der Akt. E diesen Zustand erreicht, wird ein Verhalten wie bei der StartBeforeStart-Kante ⑧ erreicht. Für rein „physische“ Aktivitäten ist dies nicht so direkt umsetzbar: So erzeugt die Akt. C aus Abb. 2b keine Daten, sondern das PMS registriert nur, dass diese Transportaktivität gestartet wurde. Die Kante ④ ist also nicht so einfach realisierbar, weil die Startbarkeit der Akt. D nicht direkt von Daten abhängig ist.

Bei Constraint-basierten Ansätzen (ein Überblick findet sich in [RW12]) wird der Kontrollfluss mittels Regeln festgelegt, welche die Menge der erlaubten Ausführungsreihenfolgen einschränken. Auch diese beziehen sich auf Aktivitäten als Ganzes, weshalb hiermit ebenfalls keine Sequenzen der Typen 2 bis 4 modelliert werden können.

[He00, He01] ermöglicht die Definition beliebiger Abhängigkeiten zwischen dem Start und dem Ende von Aktivitäten (auch die Typen 2 bis 4). Allerdings werden bei diesem Ansatz keine Abhängigkeiten zwischen Aktivitäten derselben Prozessinstanz festgelegt, sondern zwischen Aktivitäten unterschiedlicher Prozessinstanzen und sogar unterschiedlicher Vorlagen. Da sich solche Abhängigkeiten nicht auf einen einzelnen Prozessgraphen beziehen, können sie dort auch nicht als graphische Kontrollflusskonstrukte modelliert werden. Stattdessen werden sie mittels speziell hierfür erstellter regulärer Ausdrücke definiert.

In [LWR10] werden Entwurfsmuster (Pattern) vorgestellt, welche die Festlegung von minimalen und maximalen Zeitabständen zwischen Aktivitäten erlauben. Da hierbei die Start- und Endezeitpunkte der Vorgänger- und der Nachfolgeraktivitäten beliebig verwendet werden können, sind alle vier vorgestellten Kantentypen abgedeckt. Allerdings werden die zeitlichen Constraints nicht im Zusammenhang mit Kontrollfluss-Kanten betrachtet und es wird keine Ausführungssemantik für die vorgestellten Entwurfsmuster definiert.

Zusammenfassend lässt sich also feststellen, dass es zwar Arbeiten zu Zeitabständen zwischen Aktivitäten gibt, die vorgestellten Kontrollfluss-Kantentypen 2 bis 4 bisher jedoch nicht betrachtet wurden. Allerdings existieren Arbeiten aus anderen Bereichen (Abhängigkeiten zwischen GP [He00, He01], Projektmanagement [Bu18]), die eine beliebige Verwendung der Start- und Endeereignisse von Aktivitäten bei der Modellierung erlauben.

3 Verhalten eines PMS zur Ausführungszeit (Runtime)

In diesem Abschnitt wird erläutert, wie ein PMS Prozessinstanzen steuert, die erweiterte Sequenzkanten enthalten. Zuerst wird beschrieben, wie der Start bzw. die Beendigung von Aktivitäten generell verzögert oder beschleunigt werden kann. Nach der Definition einiger Grundlagen wird dann für jeden Kantentyp, die zugehörige Ausführungssemantik festgelegt.

3.1 Beeinflussung des Start- und Endezeitpunkts von Aktivitäten

Wenn eine Aktivität noch nicht gestartet werden soll (z.B. aufgrund eines zeitlichen Mindestabstands im Fall A2 in Abb. 4), dann wird noch kein entsprechender Eintrag in die Arbeitslisten der Benutzer eingestellt (wie immer bei PMS). Ist der Start früher möglich (z.B. aufgrund eines neuen Typs von Sequenzkante im Fall B1) so wird ein solcher Eintrag bereits erstellt. Muss der Start früher erfolgen (Fall B2), so „erzwingt“ dies das PMS durch Eskalationen [ARD07, Ba21a]. Bei dieser bereits heute von vielen PMS angebotenen Vorgehensweise werden Nachrichten mit einer Aufforderung zur Aktivitätenbearbeitung

(automatisch und rechtzeitig) an die potentiellen Bearbeiter, deren Vorgesetzte oder Prozessadministratoren versendet. Ebenso kann die rechtzeitige Beendigung von Aktivitäten (D2) durch Eskalationen „erzwungen“ werden. Darf eine Aktivität noch nicht beendet werden (C), so kann z.B. der Ende-Button des entsprechenden Aktivitätenprogramms deaktiviert („ausgegraut“) sein. Da dies für Benutzer irritierend sein kann, sollte zusätzlich eine gut verständliche Information angezeigt werden, warum die Beendigung dieser Aktivität noch nicht möglich ist.

Zur Realisierung einiger der in Abb. 4 dargestellten Fälle muss die Ausführungssemantik der Prozess-Engine um zusätzliche Regeln erweitert werden (z.B. für Fall B1). Diese Regeln werden im Abschnitt 3.2 erläutert. Zur Verbesserung der Übersichtlichkeit sind die Nummern solcher neuer Regeln in Abb. 4 ebenfalls angegeben. Ist für einen Fall ein zusätzlicher Ausführungszustand für Aktivitäteninstanzen erforderlich, so ist auch dieser aus Abb. 4 ersichtlich (bei A2 und C1/2).

wegen	**Start von Akt. B**		**Ende von Akt. B**	
	A: später	**B:** früher	**C:** später	**D:** früher
1: Sequenzkante	Fall existiert nicht	StartBeforeStart A B Regel 3, 8	StartBeforeEnd A EndBeforeEnd B Regel 5, 6, neuer Zustand Running-Completable	Fall existiert nicht
2: Zeit-Constraint	min. x B Regel 1, 2, 8 neuer Zustand WaitingForTime	max. x B Regel 4, durch Eskalation erzwungen	min. x B Regel 5', neuer Zustand Running-Completable	max. x B Regel 7, durch Eskalation erzwungen

Abb. 4: Übersicht über die verschiedenen Fälle

3.2 Definitionen und Ausführungsregeln

Im Folgenden werden die üblicherweise verwendeten (z.B. [RD98, We19, Ob11]) Definitionen für Geschäftsprozesse und die Menge der Ausführungszustände von Aktivitäteninstanzen so erweitert, dass sie als Grundlage für die Festlegung von Ausführungsregeln für erweiterte Sequenzkanten geeignet sind.

Def. 1: Prozesstemplate, Prozessinstanz und Aktivitätenzustände
Ein Prozesstemplate PT=(N, C, T) besteht aus einer Menge von Knoten N (engl.: Node), einer Menge von Kontrollflusskanten C (Control Flow) und einer Menge von Zeitkanten T (Time).
Eine Prozessinstanz PI=(PT, State) besteht aus einem Prozesstemplate PT und einer Menge von Aktivitätenzuständen (State). Jede Aktivität a $\in$ N kann einen unterschiedlichen Zustand State(a) haben. Aktivitäten können die in Abb. 6 dargestellten Zustände einnehmen, d.h.
$\forall$ a $\in$ N gilt: State(a) $\in$ {Inactive, WaitingForTime, Active, Running, RunningCompletable, Completed}

Das PMS kennt den aktuellen Zustand jeder Aktivität und verändert diese Zustände während der Ausführung der Prozessinstanz mittels vorgegebener Ausführungsregeln. Dadurch kann das PMS die Ausführungsreihenfolge der Aktivitäten steuern, Einträge in Arbeitslisten einfügen, Aktivitätenprogramme (z.B. Formulare) starten, automatisch ausgeführte Services aufrufen, etc.

Def. 2: Eine Kontrollflusskante c ∈ C ist definiert als
c = (SourceAct, TargetAct, Type) mit:
SourceAct: die Quellaktivität der Kante
TargetAct: die Zielaktivität der Kante
Type: der Typ der Kante mit
Type ∈ {StartBeforeStart, StartBeforeEnd, EndBeforeStart, EndBeforeEnd}

Wie in Def. 1 erkennbar ist, stellen Kontrollfluss- und Zeitkanten unterschiedliche Kantentypen dar. Der Grund für diese Trennung ist, dass die Existenz eine Zeitkante zwischen zwei Aktivitäten nicht immer auch eine Sequenz-Abhängigkeit impliziert: Wie in Abb. 5 dargestellt, kann gefordert sein, dass der Start einer Akt. B spätestens (d.h. max.) 10 Stunden nach dem Ende der Akt. A erfolgen muss (d.h. klassischer Typ EndBeforeStart). Um den Ablauf zusätzlich zu beschleunigen, kann aber erlaubt sein, dass der Start der Akt. B bereits vor der Beendigung der Akt. A erfolgen darf, d.h. es existiert zwischen den beiden Aktivitäten keine Kontrollflusskante dieses Typs EndBeforeStart, sondern des Typs StartBeforeStart (vgl. Abb. 5).

Abb. 5: Aktivitäten mit Zeitkante, aber ohne „klassische“ Kontrollflusskante

Def. 3: Eine Zeitkante t ∈ T ist definiert als
t = (SourceAct, TargetAct, Type, MinTime, MaxTime) mit:
SourceAct, TargetAct und Type: wie in Def. 2 festgelegt
MinTime: die für die Kante festgelegte minimale Zeitdauer
MaxTime: die für die Kante festgelegte maximale Zeitdauer,
wobei bei Kanten, für die kein entsprechendes Zeit-Constraint definiert wurde,
MinTime bzw. MaxTime den Wert undef haben

Um diese Informationen in den Ausführungsregeln nutzen zu können, kann mit gleichnamigen Funktionen auf sie zugegriffen werden. So liefert z.B. SourceAct(c) die Quellaktivität der Kante c und State(a) den aktuellen Ausführungszustand der Akt. a.

Die Menge der Zustände einer Aktivitäteninstanz wurde erweitert, um die neuen Typen von Sequenzkanten realisieren zu können. Die beiden zusätzlich erforderlichen Zustände sind in Abb. 6 farbig hinterlegt. Außerdem sind die Nummern der für erweiterte Sequenzkanten zusätzlich erforderlichen Regeln den entsprechenden Kanten zugeordnet. Die aus klassischen PMS bereits bekannten Regeln (vgl. [RD98, We19, Ob11]) werden im Folgenden nicht wiederholt, weswegen in Abb. 6 auch Kanten ohne Beschriftung existieren. Auf das Verhalten

bei Verzweigungen, Parallelitäten und Schleifen wird nur kurz eingegangen. Stattdessen stehen erweiterte Sequenzkanten im Fokus. Außerdem werden keine Zustände betrachtet, die hierfür nicht direkt relevant sind. Solche Zustände sind z.B. für einen Abbruch einer Aktivität, deren Kompensation, erfolglos beendete Aktivitäten (Zustand Failed in [Ob11]) oder bei Sprüngen zu anderen Aktivitäten einer Prozessinstanz [Ba18] erforderlich.

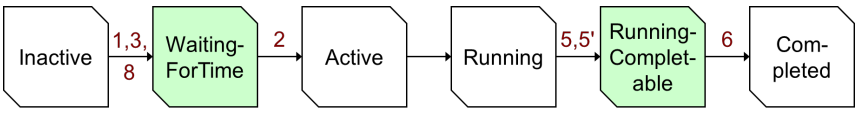

Abb. 6: Zustandsübergangsdiagramm für Aktivitäteninstanzen

Späterer Start einer Aktivität - Sequenzkante: Es kann nicht vorkommen, dass eine Akt. B durch einen der neuen Sequenzkanten-Typen später startbar ist, als bei klassischen Sequenzkanten: Der einzig neue Typ, der sich auf den Start einer Zielakt. B bezieht, ist StartBeforeStart. Existiert eine Kante dieses Typs z.B. von Akt. A zu Akt. B (Fall B1 in Abb. 4), so ist die Akt. B jedoch früher startbar als bei einer klassischen Kante (Typ EndBeforeStart) – und nicht später.

Zeitkante: Eine Akt. B kann später startbar sein, weil sie die Zielaktivität einer Zeitkante mit vorgegebener Minimalzeit ist (A2 in Abb. 4). Da die Verzögerung den Start der Aktivität betrifft, kann die Kante den Typ EndBeforeStart oder StartBeforeStart haben.

Damit das PMS solche Zeitkanten zur Runtime verarbeiten kann, ist der zusätzliche Aktivitätenzustand WaitingForTime erforderlich (vgl. Abb. 6): Dass die Vorgängeraktivität von Akt. B bereits abgeschlossen ist, „merkt“ sich die Prozess-Engine, indem Akt. B den Startzustand Inactive verlässt. Würden ausschließlich Kontrollflusskanten berücksichtigt werden, so wäre die Akt. B bereits startbar, d.h. bei den „klassischen“ Ausführungsregel würde sie in den Zustand Active wechseln. Dies ist jedoch aufgrund des Zeit-Constraints noch nicht erlaubt, weswegen dieser Fall durch den neuen Zustand WaitingForTime unterschieden wird. Die „klassische“ Ausführungsregel wird so verändert, dass auf Inactive nicht direkt in der Zustand Active folgt. Stattdessen erfolgt zuerst ein Wechsel in den Zustand WaitingForTime:

Regel 1: Die Kante $c \in C$ sei eine in die Aktivität a mündende „normale“ Kontrollflusskante, d.h. TargetAct(c) = a $\wedge$ Type(c) = EndBeforeStart. Nach Beendigung der Quellaktivität q dieser Kante, wechselt die Akt. a in den Zustand WaitingForTime:
Wird bei der Akt. q = SourceAct(c) der State(q) = Completed erreicht, dann ändert sich der Zustand von Akt. a: State(a) = WaitingForTime

Ein Wechsel vom Zustand WaitingForTime in den Zustand Active erfolgt mittels der Ausführungsregel 2, sobald alle minimalen Wartezeiten der eingehenden Zeitkanten erreicht sind, so dass die Akt. a nun tatsächlich bearbeitet werden darf.

Regel 2: T_a sei die Menge der für die Berechnung des frühesten Startzeitpunkts relevanten Zeitkanten einer Akt. a mit State(a) = WaitingForTime:

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

Dann ergibt sich die früheste Startzeit $EarliestStartingTime_a$ der Akt. a als der größte (d.h. späteste) Zeitpunkt, der aus einer dieser Kanten resultiert:

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

Hierfür wird der aus der Kante t resultierende Zeitpunkt $ResultingTime_{Min,Start}(t)$ berechnet, indem (abhängig vom Kantentyp) zur Start- bzw. Endezeit der Quellaktivität der Kante t die für die Kante t festgelegte Minimalzeit addiert wird:[2]

$ResultingTime_{Min,Start}(t_i) =$

$$\begin{cases} StartTime(SourceAct(t_i)) + MinTime(t_i), \text{ falls } Type(t_i) = StartBeforeStart \\ EndTime(SourceAct(t_i)) + MinTime(t_i), \text{ falls } Type(t_i) = EndBeforeStart \end{cases}$$

Sobald die Zeit $EarliestStartingTime_a$ erreicht ist (d.h. $CurrentTime \geq EarliestStartingTime_a$) wird der Zustand von Akt. a in Active geändert: State(a) = Active.

Bemerkung: Falls es keine solchen Zeitkanten gibt (d.h. $T_a = \{\}$), ergibt die Berechnung des Maximums (Max) eine $EarliestStartingTime_a = -\infty$. Da stets $CurrentTime \geq -\infty$ gilt, wird direkt in State(a) = Active gewechselt, sobald State(a) = WaitingForTime erreicht ist. Der Zustand WaitingForTime ist dann also nicht relevant und wird sofort wieder verlassen.

Die nachfolgend erläuterten Varianten von ResultingTime werden später in den Ausführungsregeln 4, 5′ und 7 benötigt: $ResultingTime_{Max,Start}(t_i)$ wird wie in Regel 2 berechnet, wobei jedoch $\underline{Max}Time(t_i)$ (anstatt $MinTime(t_i)$) addiert wird. Außerdem werden $ResultingTime_{Min,End}(t_i)$ und $ResultingTime_{Max,End}(t_i)$ auf dieselbe Weise berechnet, wobei jedoch die Menge T_a Kanten t_i mit $Type(t_i) \in \{StartBefore\underline{End}, EndBefore\underline{End}\}$ enthält.

Früherer Start einer Aktivität - Sequenzkante: Außer durch klassische Sequenzkanten (d.h. Typ EndBeforeStart) kann eine Akt. B auch durch eine Kante vom Typ StartBeforeStart ausführbar werden (Fall B1 in Abb. 4). Auch dann muss ihr Zustand von Inactive zu WaitingForTime wechseln (noch nicht Active, wie im vorherigen Abschnitt erläutert). Dies wird durch die folgende Ausführungsregel realisiert:

Regel 3: Die Akt. a mit State(a) = Inactive sei das Ziel der Kontrollflusskante $c \in C$ vom Typ StartBeforeStart, d.h. $TargetAct(c) = a \wedge Type(c) = StartBeforeStart$.
Nach Start der Quellakt. q dieser Kante, wechselt die Akt. a in den Zustand WaitingForTime: Wird bei der Akt. q = SourceAct(c) der State(q) = Running gesetzt, dann ändert sich der Zustand von Akt. a: State(a) = WaitingForTime

Zeitkante: Hat eine Akt. B eine eingehende Zeitkante mit einer Maximalzeit (B2 in Abb. 4), so muss ihr Start bis zu einem gewissen Zeitpunkt ($LatestStartingTime_a$, s.u.) durch Eskalationen erzwungen werden. Dabei sind nur Zeitkanten relevant, die sich auf den Start der Akt. B beziehen (also Typen ..BeforeStart). Der Zeitpunkt $LatestStartingTime_a$ berechnet sich wie folgt:

[2] StartTime(a) einer Akt. a ist der Zeitpunkt, an dem diese Aktivität gestartet wurde. EndTime(a) ist der Zeitpunkt ihrer Beendigung. CurrentTime ist der aktuelle Zeitpunkt.

Regel 4: T_a sei die Menge der für die Berechnung des spätesten Startzeitpunkts von Akt. a relevanten Zeitkanten:
$T_a = \{t_i \in T \mid TargetAct(t_i) = a \wedge Type(t_i) \in \{StartBeforeStart, EndBeforeStart\} \wedge MaxTime(t_i) \neq undef\}$
Dann ergibt sich die späteste Startzeit $LatestStartingTime_a$ der Akt. a als der kleinste (d.h. früheste) Startzeitpunkt, der aus einer dieser Kanten t_i als späteste (d.h. Max) Startzeit resultiert:
$LatestStartingTime_a = Min(ResultingTime_{Max,Start}(t_i)) \; \forall \; t_i \in T_a$

Wie bereits erwähnt, soll dieser späteste Startzeitpunkt z.B. durch Eskalationen sichergestellt werden. Da nach der Auslösung einer Eskalation eine gewisse Zeit vergeht, bis diese vom Benutzer erkannt und die Aktivität auch tatsächlich gestartet wird, sollte die Eskalation rechtzeitig vor Erreichen der $LatestStartingTime_a$ ausgelöst werden. In gewissen Anwendungsfällen können auch mehrstufige Eskalationen [ARD07] sinnvoll sein: Beispielsweise kann zuerst eine eMail an die potentiellen Bearbeiter der Aktivität gesendet werden. Wurde die Aktivität nach einer gewissen Zeit immer noch nicht gestartet, so wird ein Prozessverantwortlicher (Administrator, Vorgesetzter) informiert. Auch dies muss noch rechtzeitig vor Erreichen von $LatestStartingTime_a$ erfolgen.

Späteres Ende einer Aktivität - Sequenzkante: Durch die für den Fall C1 in Abb. 4 dargestellten Typen (..BeforeEnd) von Sequenzkanten, kann die Beendigung der Akt. B verzögert werden. Das bedeutet, dass der Benutzer die Akt. B evtl. bereits bearbeitet hat, diese aufgrund des vorgegebenen Prozesstemplates aber noch nicht beenden darf. Damit die Prozess-Engine diesen Fall erkennen kann, wird der neue Zustand RunningCompletable eingeführt. Die Akt. B befindet sich während ihrer Bearbeitung zuerst im Zustand Running. Erst wenn die mit einer solchen Kante verbundene Vorgängerakt. A gestartet bzw. beendet wurde, darf Akt. B abgeschlossen werden, weshalb sich ihr Zustand zu RunningCompletable ändert. Die nun folgende Regel 5 realisiert diesen Zustandsübergang.

Regel 5: C_a sei die Menge der für die Beendigung der Akt. a mit State(a) = Running relevanten Kontrollflusskanten:
$C_a = \{c_i \in C \mid TargetAct(c_i) = a \wedge Type(c_i) \in \{StartBeforeEnd, EndBeforeEnd\}\}$
Ein Zustandswechsel der Akt. a ist erlaubt, sobald alle Quellaktivitäten dieser Kanten gestartet (i) bzw. beendet (ii) wurden, d.h. alle folgenden Bedingungen erfüllt sind:
i) Falls $\forall c_i \in C_a$ mit $Type(c_i) = StartBeforeEnd$ gilt:
$State(SourceAct(c_i)) \in \{Running, RunningCompletable, Completed\}$ und
ii) falls $\forall c_i \in C_a$ mit $Type(c_i) = EndBeforeEnd$ gilt:
$State(SourceAct(c_i)) = Completed$,
dann wechselt der Zustand von Akt. a in: State(a) = RunningCompletable

Bemerkung: Wenn es keine Kontrollflusskanten dieses Typs gibt, deren Zielaktivität die Akt. a ist (d.h. $C_a = \{\}$), dann sind die Bedingungen i) und ii) automatisch erfüllt und der Zustand geht unmittelbar nach Erreichen von Running in RunningCompletable über. Der

Grund dafür ist, dass die Bedingung „für alle“ ($\forall$) c_i erfüllt ist, wenn es gar keine solchen c_i gibt.

Die nachfolgende Regel 6 ersetzt die klassische bei der Beendigung einer Aktivität ausgeführte Regel. Die wesentliche Änderung ist, dass Regel 6 den (neuen) Zustand RunningCompletable verwendet, anstatt den Zustand Running.

Regel 6: Befindet sich eine Akt. a im Zustand State(a) = RunningCompletable, dann darf der Benutzer diese Aktivität abschließen. Dadurch ändert sich ihr Zustand in:
State(a) = Completed

Wenn eine Aktivität noch nicht beendet werden darf (d.h. noch im Zustand Running anstatt RunningCompletable ist), dann kann dies dem Benutzer signalisiert werden, indem z.B. der Button „Aktivitätenprogramm beenden“ oder ein entsprechender Eintrag im Menü deaktiviert ist. Zusätzlich sollte ihm (zuvor im Prozesstemplate modellierte) Information angezeigt werden, warum eine Beendigung noch nicht möglich ist. Damit ein Aktivitätenprogramm entsprechend gestaltet werden kann, muss die Schnittstelle (API) des PMS Funktionen anbieten, um solche Hilfetexte abzufragen und um zu ermitteln, ob eine bestimmte Aktivität bereits beendet werden darf. Die Funktion zum Beenden der Akt. a soll einen Fehler zurückliefern, wenn dies bereits im Zustand State(a) = Running versucht wird.

Zeitkante: Außer durch die eben betrachteten Kontrollflusskanten kann auch durch Zeitkanten mit vorgegebener Minimalzeit die Beendigung der Akt. B verzögert werden (vgl. C2 in Abb. 4). Da sie sich auf das Ende der Akt. B beziehen sollen, muss es sich um Zeitkanten der Typen ..BeforeEnd handeln. Um diese Kanten zu berücksichtigen, wird die oben dargestellte Regel 5 um eine weitere Bedingung (iii) erweitert, die ebenfalls erfüllt sein muss:

Regel 5′: T_a sei die Menge der für die Berechnung des frühesten Endezeitpunkts relevanten Zeitkanten der Akt. a (mit State(a) = Running):
$T_a = \{t_i \in T \mid TargetAct(t_i) = a \wedge Type(t_i) \in \{StartBeforeEnd, EndBeforeEnd\} \wedge$
$MinTime(t_i) \neq undef \}$
Der früheste erlaubte Zeitpunkt für die Beendigung der Akt. a ist dann:
$EarliestCompletionTime_a = Max(ResultingTime_{Min,End}(t_i)) \; \forall \; t_i \in T_a$
iii) Ein Zustandswechsel der Akt. a ist erlaubt, sobald gilt:
$CurrentTime \geq EarliestCompletionTime_a$

Für einen Zustandswechsel müssen also alle 3 Bedingungen erfüllt sein, d.h. i) und ii) aus Regel 5 und iii) aus Regel 5′. Wenn allerdings keine entsprechenden Zeitkanten existieren (d.h. $T_a = \{\}$), dann ergibt die Berechnung des Maximums wieder den Wert $-\infty$ und die $\geq$-Bedingung für CurrentTime ist stets erfüllt, und damit auch Bedingung iii).

Es ist auch erlaubt, dass eine Zeitkante $t_i \in T_a$ vom Typ StartBeforeEnd die Akt. a sowohl als Quell- wie auch als Zielaktivität verwendet. Dies stellt den durchaus relevanten Sonderfall dar, dass für eine Aktivität eine Mindest-Bearbeitungsdauer festgelegt wird,

z.B. die Zeitdauer zum Trocknen einer Verklebung. Dieser Sonderfall ist in Regel 5′ auch eingeschlossen.

Früheres Ende einer Aktivität - Sequenzkante: Es kann nicht vorkommen, dass durch die neuen Typen von Kontrollflusskanten eine Akt. a früher beendet werden kann, als in klassischen Prozessmodellen. Bei letzteren kann die Akt. a unmittelbar nach ihrem Start beendet werden. Eine frühere Beendigung (d.h. vor dem Starten) macht keinen Sinn und kann deshalb auch nicht aus den neuen Kontrollflusskanten-Typen resultieren.

Zeitkante: Durch Zeitkanten mit einer Maximal-Zeit kann gefordert sein, dass eine Akt. B bis zu einem bestimmten Zeitpunkt beendet wird (vgl. D2 in Abb. 4). Dabei muss es sich um Kanten handeln, die sich auf die Beendigung der Akt. B beziehen (d.h. die Typen ..BeforeEnd) und der Beendigungszeitpunkt wird vom PMS wieder durch Eskalationen sichergestellt. Die Regel 7 dient zur Berechnung dieses Zeitpunkts.

Regel 7: T_a sei die Menge der für die Berechnung des spätesten Endezeitpunkts von Akt. a relevanten Zeitkanten:
$T_a = \{t_i \in T \mid TargetAct(t_i) = a \wedge Type(t_i) \in \{StartBeforeEnd, EndBeforeEnd\} \wedge MaxTime(t_i) \neq undef\}$
Dann ergibt sich der späteste Endezeitpunkt $LatestCompletionTime_a$ der Akt. a als der kleinste (d.h. früheste) Maximal-Endezeitpunkt, der aus einer dieser Kanten t_i resultiert:
$LatestCompletionTime_a = Min(ResultingTime_{Max,End}(t_i)) \; \forall \; t_i \in T_a$

Gateways: Im Folgenden wird speziell auf das Verhalten bei Gateways eingegangen, da dies durch die bisher vorgestellten Regeln nicht vollständig abgedeckt ist. Für nach einem Split liegende Aktivitäten (z.B. die Akt. B, C und E in Abb. 2b) sind keine Erweiterungen notwendig, da diese nur eine einzige Vorgängeraktivität haben (die Split-Aktivität A in Abb. 2b). Außerdem decken die vorgestellten Regeln bereits den Fall mehrerer eingehender Kanten der Typen StartBeforeEnd und EndBeforeEnd (beide durch Regel 5) sowie mehrere Zeitkanten (Regeln 2, 4, 5′ und 7) ab.

Um die Lesbarkeit zu verbessern, wurde bei den Regeln 1 und 3 bisher der Fall mehrerer eingehender Kanten der Typen EndBeforeStart sowie StartBeforeStart ausgeklammert. Diese treten z.B. bei AND-Joins[3] auf (Akt. F in Abb. 2b)[4]. Bei diesen muss auf alle Vorgängeraktivitäten gewartet werden. Hierzu werden die Regeln 1 und 3 durch die nachfolgend dargestellte Regel 8 ersetzt, bei der die Menge Q_{End} alle mit Kanten des Typs EndBeforeStart und Q_{Start} mit StartBeforeStart verbundene Vorgängeraktivitäten enthält.

[3] Auch OR- und XOR-Join-Knoten haben mehrere eingehende Kanten. Bei diesen sind nur diejenigen Vorgängeraktivitäten des Join-Knotens zu berücksichtigen, die in bei dieser Prozessinstanz tatsächlich ausgeführten Pfaden liegen. Die Mengen Q_{End} und Q_{Start} enthalten also keine Aktivitäten aus nicht ausgeführten Pfaden.

[4] Im Gegensatz zu BPMN verzichten wir Abb. 2b auf separate Gateway-Knoten (ebenso wie z.B. ADEPT [RD98, Re00]), weil Gateways keine nennenswerte Zeit zum „Durchschalten“ benötigen. Deswegen fallen ihr Start- und Endeereignis quasi zusammen, so dass der separate Knoten im betrachteten Kontext ebenso irrelevant wäre, wie die zusätzlich entstehende Kante. (In Abb. 2b ergäbe sich ein Join-Knoten mit den eingehenden Kanten ⑤, ⑥ und ⑧, sowie eine zusätzliche ausgehende Kante zur Akt. F, deren Typ End/StartBeforeStart irrelevant ist, weil die entsprechenden Ereignisse fast gleichzeitig stattfinden.)

Neu ist, dass nicht eine einzelne Vorgängeraktivität q den erforderlichen Zustand erreichen muss, sondern alle Aktivitäten dieser beiden Mengen. So ergeben sich bei Abb. 2b für die Akt. F die Mengen $Q_{End} = \{C, D\}$ und $Q_{Start} = \{E\}$ (letztere wegen der Kante ⑧, die Kante ⑨ spielt für die Startbarkeit der Akt. F keine Rolle).

Regel 8: C_{End} sei die Menge der in Aktivität a mündenden „normalen" Kontrollflusskanten:
$C_{End} = \{c_i \in C \mid TargetAct(c_i) = a \wedge Type(c_i) = EndBeforeStart\}$
C_{Start} sei die Menge der in Aktivität a mündenden Kanten des Typs StartBeforeStart:
$C_{Start} = \{c_i \in C \mid TargetAct(c_i) = a \wedge Type(c_i) = StartBeforeStart\}$
Nach Beendigung aller zugehöriger Quellaktivitäten $q \in Q_{End}$ der Kanten aus C_{End} und nach dem Starten aller Aktivitäten $q \in Q_{Start}$ der Kanten aus C_{Start} wechselt die Akt. a in den Zustand WaitingForTime:
Wenn $\forall\, q \in Q_{End}$ mit $Q_{End} = \{q \in N \mid \exists\, c_i \in C_{End} \wedge q = SourceAct(c_i)\}$ gilt State(q) = Completed und $\forall\, q \in Q_{Start}$ mit $Q_{Start} = \{q \in N \mid \exists\, c_i \in C_{Start} \wedge q = SourceAct(c_i)\}$ gilt State(q) = Running, dann ändert sich der Zustand von Akt. a: State(a) = WaitingForTime

4 Zusammenfassung und Ausblick

Der vorgestellte Ansatz erweitert Sequenzkanten, indem sich diese nun beliebig auf die Start- und Endeereignisse ihrer Quell- und Zielaktivitäten beziehen dürfen. Außerdem wird die Festlegung von minimalen und maximalen Zeitabständen ermöglicht, die sich ebenfalls auf diese Ereignisse beziehen. Dieser Beitrag erläutert, wie ein PMS (z.B. durch Eskalationen) die Benutzer so beeinflussen kann, dass all diese modellierten Bedingungen eingehalten werden. Außerdem wird die formale Ausführungssemantik von Prozess-Engines erweitert, indem zusätzlich erforderliche Zustände für Aktivitäteninstanzen eingeführt und neue Ausführungsregeln definiert werden. Damit wird es einem PMS ermöglicht, solche GP automatisch zu steuern, die die eben erwähnten Konstrukte enthalten.

Die vorgestellten Regeln müssen mittels einer prototypischen Implementierung noch technisch evaluiert werden. Ideal wäre hierfür deren Integration in ein existierendes und bereits praktisch nutzbares PMS, weil dies zusätzlich eine (fachliche) Evaluation ihrer Eignung für GP-Modellierer und Endanwender ermöglicht. Allerdings ist eine solche Integration sehr aufwendig und, wegen der Komplexität von Prozess-Engines, eigentlich nur durch den jeweiligen Hersteller realisierbar. Andererseits ist eben diese Erweiterung existierender PMS um die hier beschriebenen Funktionalitäten das längerfristige Ziel, weil diese so einem großen Nutzerkreis bereitgestellt werden können. Selbst deren Integration in GP-Modellierungswerkzeuge zur reinen (fachlichen / semantischen) GP-Dokumentation und -Optimierung ist erstrebenswert, weil sie eine detailliertere GP-Modellierung ermöglichen. Durch Analyse der dann entstehenden GP-Modelle kann später ermittelt werden, wie häufig die vorgestellten Konstrukte in realen GP benötigt werden.

Literaturverzeichnis

[Al83] Allen, J.F.: Maintaining knowledge about temporal intervals. Communications of the ACM, 26(11):832–843, 1983.

[ARD07] Aalst, W.M.P. van der; Rosemann, M.; Dumas, M.: Deadline-based Escalation in Process-Aware Information Systems. Decision Support Systems, 43(2):492–511, 2007.

[AWG05] Aalst, W.M.P. van der; Weske, M.; Grünbauer, D.: Case Handling: A New Paradigm for Business Process Support. Data & Knowledge Engineering, 53(2):129–162, 2005.

[Ba18] Bauer, T.: Ausführungssemantik für Sprünge in Geschäftsprozessen. Datenbank-Spektrum, 18(2):99–111, 2018.

[Ba19a] Bauer, T.: Modellierung erweiterter Beziehungen zwischen dem Start und dem Ende von Aktivitäten in Geschäftsprozessen: Szenarien, Anforderungen und Darstellungsvarianten. In: Proc. Informatik 2019, Gemeinsamer Workshop IT-Governance und Strategisches Informationsmanagement. S. 353–366, 2019.

[Ba19b] Bauer, T.: Pre-modelled Flexibility for Business Processes. In: Proc. 21th Int. Conf. on Enterprise Information Systems. S. 547–555, 2019.

[Ba21a] Bauer, T.: Effiziente Modellierung von Eskalationen und Stellvertretungen in Geschäftsprozessen. In: Proc. Informatik 2021, Workshop zum Stand, den Herausforderungen und Impulsen des Geschäftsprozessmanagements. S. 1503–1516, 2021.

[Ba21b] Bauer, T.: Pre-modelled Flexibility for the Control-Flow of Business Processes: Requirements and Interaction with Users. In: Enterprise Information Systems. S. 833–857, 2021.

[Bu18] Burghardt, M.: Projektmanagement: Leitfaden für die Planung, Überwachung und Steuerung von Projekten. Publicis Publishing, Erlangen, 10. Auflage, 2018.

[Da18] Dakic, D.; Stefanovic, D.; Cosic, I.; Lolic, T.; Medojevic, M.: Business Process Mining Application: A Literature Review. In: Proc. 29th DAAAM International Symposium on Intelligent Manufacturing and Automation. S. 866–875, 2018.

[He00] Heinlein, C.: Workflow- und Prozeßsynchronisation mit Interaktionsausdrücken und -graphen: Konzeption und Realisierung eines Formalismus zur Spezifikation und Implementierung von Synchronisationsbedingungen. Dissertation, Universität Ulm, Fakultät für Informatik, 2000.

[He01] Heinlein, C.: Workflow and Process Synchronization with Interaction Expressions and Graphs. In: Proc. 17th Int. Conf. on Data Engineering. S. 243–252, 2001.

[HW16] Hewelt, M.; Weske, M.: A Hybrid Approach for Flexible Case Modeling and Execution. In: Proc. 14th Int. Conf. on Business Process Management, Business Process Management Forum. S. 38–54, 2016.

[LWR10] Lanz, A.; Weber, B.; Reichert, M.: Workflow Time Patterns for Process-Aware Information Systems. In: Proc. Enterprise, Business-Process, and Information. S. 94–107, 2010.

[OA07] OASIS: Web Services Business Process Execution Language Version 2.0. Standard, 2007. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf.

[Ob11] Object Management Group: Business Process Model and Notation (BPMN) 2.0. Standard, 2011. http://www.omg.org/spec/BPMN/2.0.

[RD98] Reichert, M.; Dadam, P.: ADEPTflex - Supporting Dynamic Changes of Workflows Without Losing Control. Journal of Intelligent Information Systems, Special Issue on Workflow Management Systems, 10(2):93–129, 1998.

[Re00] Reichert, M.: Dynamische Ablaufänderungen in Workflow-Management-Systemen. Dissertation, Universität Ulm, Fakultät für Informatik, 2000.

[RH06] Russell, N.; Hofstede, A.H.M. ter: Workflow Control-Flow Patterns: A Revised View. BPM Center Report BPM-06-22, 2006.

[RW12] Reichert, M.; Weber, B.: Enabling Flexibility in Process-Aware Information Systems: Challenges, Methods, Technologies. Springer, 2012.

[We19] Weske, M.: Business Process Management: Concepts, Languages, Architectures. Springer, 3. Auflage, 2019.

[ZSA21] Zerbino, P.; Stefanini, A.; Aloini, D.: Process Science in Action: A Literature Review on Process Mining in Business Management. Technological Forecasting and Social Change, 172, 2021.