Maximilian Eibl, Martin Gaedke. (Hrsg.): INFORMATIK 2017,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2017 799

Anforderungen an vormodellierte Flexibilität für den Kontrollfluss von Geschäftsprozessen

Thomas Bauer[1]

Abstract: Bei Process-aware Information Systems (PAIS) ist manchmal ein flexibles Abweichen vom starr modellierten Prozess notwendig. Ansonsten werden die Benutzer zu stark eingeschränkt. Deshalb wurde im Projekt CoPMoF ein Ansatz entwickelt, bei dem zu erwartende, benötigte Flexibilität bereits zur Buildtime einmalig vormodelliert wird. Dies hat gegenüber dynamischen Änderungen zur Runtime den Vorteil, dass, durch Nutzung der vormodellierten Informationen, bei jeder Abweichung zur Runtime viel weniger Aufwand für die Benutzer entsteht. Außerdem wird die Anwendung der Flexibilität sicherer, z.B. weil hierfür Benutzerrechte definiert werden können. In diesem Beitrag werden die entsprechenden Anforderungen dargestellt, wobei insbesondere darauf eingegangen wird, welche Informationen zur Buildtime vormodelliert werden müssen. Außerdem wird die Notwendigkeit der Anforderungen an Praxisbeispielen erläutert.

Keywords: Process-aware Information System, Prozess-Engine, Buildtime, Flexibilität

1 Einleitung

Ein Vorteil von PAIS [RW12] gegenüber klassischen IT-Systemen ist, dass die Einhaltung des vorgegebenen Prozesses durch das Prozess-Management-System (PMS) sichergestellt wird (Prozesssicherheit). Außerdem werden die Anwender von nicht-produktiven Aufgaben entlastet, wie dem Suchen der richtigen Programmfunktion oder der im aktuellen Geschäftsvorfall benötigten Daten. Dies erfolgt bei einem PAIS automatisch. Allerdings haben PAIS auch Nachteile: So können sich Benutzer durch die aktive Steuerung gegängelt fühlen. Außerdem kann die Einschränkung der möglichen Ausführungsreihenfolgen der Prozessaktivitäten dazu führen, dass in Sonderfällen bestimmte Reihenfolgen nicht realisierbar sind, obwohl hierfür eine betriebliche Notwendigkeit besteht. Dadurch entsteht ein Nachteil für das Unternehmen.

Um solche Nachteile zu vermeiden, muss flexibel vom modellierten Geschäftsprozess abgewichen werden können [Re09, Sc07]. Eine spezielle Art von Flexibilität sind Abweichungen, die bereits zur Modellierungszeit (Buildtime) vormodelliert werden, damit sie zur Ausführungszeit (Runtime) der Prozessinstanzen angewandt werden können (Pre-Designed Flexibility [KN06], Flexibility by Design [Sc07]). In der wissenschaftlichen Literatur findet sich zu diesem Thema jedoch fast ausschließlich diese Kategorisierung. Eine Detaillierung der entsprechenden Anforderungen und von Ansätzen

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

doi:10.18420/in2017_80

zu deren Umsetzung waren bisher kaum Inhalt von Forschungsarbeiten.

Dieser Aspekt ist der Fokus des Projekts CoPMoF (Controlable Pre-Modeled Flexibility). Die Flexibilität in PMS soll vergrößert werden, aber Veränderungen sollen nicht willkürlich (d.h. voll dynamisch) durch die Benutzer festgelegt werden. Stattdessen wird vorhersehbare, möglicherweise zur Runtime benötigte Flexibilität bereits zur Buildtime vormodelliert. Solche Abweichungen können vom Geschäftsprozess-Modellierer (bzgl. ihrer Auswirkungen) gründlich durchdacht und ggf. vom Prozessverantwortlichen genehmigt werden. Dadurch ist die notwendige Prozesssicherheit weiterhin gewährleistet und Abweichungen sind nur im vorgesehenen Umfang und nur durch Benutzer mit entsprechenden Rechten möglich. Der entscheidende Vorteil ist jedoch, dass, für die Auslösung einer Abweichung, für die Benutzer ein sehr viel geringerer Aufwand entsteht, als für eine dynamische Änderung (evtl. wären sie mit deren Definition sogar überfordert). Angenommen, eine telefonische Rückfrage beim Kunden schlägt fehl. Jetzt könnte eine Aktivität „Nachfrage per Post“ dynamisch in den Geschäftsprozess eingefügt werden. Dazu müssten jedoch alle nachfolgend beschriebenen Festlegungen vom Benutzer des PAIS getroffen werden, die besser zur Buildtime vormodelliert werden sollten. Dann wurde nämlich bereits festgelegt, an welcher Stelle im Kontrollfluss diese zusätzliche Aktivität platziert werden soll (d.h. die Vorgänger- und Nachfolgeraktivitäten festgelegt). Auch der Datenfluss wurde bereits definiert, d.h. die Anbindung von Aktivitäten-Input- und -Output-Parametern an Prozessvariablen. So bezieht z.B. der Input-Parameter Adresse im Feld Straße seinen Inhalt von der Prozessvariablen CustomerAddress und dort vom Attribut CustomerStreet. Ebenso wurde bereits eine geeignete Bearbeiterzuordnung vorgegeben, z.B. „Rolle = Kreditbearbeiter und Abteilung = x“, wobei sich x aus der Prozessvariablen CreditApplication und dort dem Attribut ExecutingUnit ergibt.

Dynamische Änderungen [RD98, RW12] ermöglichen z.B. das Einfügen von (in der Prozessvorlage nicht vorgesehenen) Aktivitäten für eine einzelne Prozessinstanz. Ebenso wird das Löschen oder Verschieben von Aktivitäten ermöglicht. Zur Umsetzung von nicht vorhersehbaren Änderungen stellen diese eine unverzichtbare Funktionalität dar. Für vorhersehbare Ausnahmesituationen und Sonderfälle ist dieser Mechanismus jedoch weniger geeignet, weil (wie bereits erläutert) bei jeder Abweichung ein Mehraufwand für die Benutzer entsteht. Hier ist es deutlich sinnvoller, den Aufwand ein einziges Mal bereits zur Buildtime zu betreiben, und die evtl. benötigte Flexibilität vorzumodellieren.

Bzgl. der Vormodellierung von Flexibilität wird in der wissenschaftlichen Literatur (wie erwähnt) bisher lediglich die entsprechende Kategorie von Flexibilität definiert, diese Kategorie wird jedoch nicht näher untersucht. Damit verbleibt folgende bisher unbeantwortete Forschungsfrage: Welche Szenarien (d.h. Anforderungen) existieren, bei denen es vorteilhaft ist, zur Buildtime ein flexibles Verhalten vorzumodellieren, und welche Informationen müssen hierbei jeweils vorgegeben werden?

In diesem Beitrag werden zahlreiche unterschiedliche Szenarien für vormodellierte Flexibilität detailliert dargestellt. Dabei wird auf diverse Einzelaspekte und -anforderungen eingegangen, um die Szenarien umfassend und verständlich darzustellen. Es wird

ausschließlich der Kontrollfluss betrachtet (für weitere Prozessaspekte siehe [Ba17]). Außerdem wird die Notwendigkeit der Anforderungen an Praxisbeispielen erläutert.

Im nachfolgenden Abschnitt werden einige Begriffe eingeführt und die prinzipielle Funktionsweise von PAIS erläutert. Außerdem wird die Problemstellung an einem Anwendungsbeispiel motiviert, das später wieder aufgegriffen wird. In Abschnitt 3 werden die einzelnen Anforderungen erläutert und Praxisbeispiele hierzu dargestellt. Abschnitt 4 stellt verwandte Arbeiten vor.

2 Grundlagen und resultierende Problemstellungen

Im Folgenden wird beschrieben, welche Art vormodellierter Flexibilität heute üblicherweise bereits in den meisten Ansätzen und kommerziellen Systemen zur Prozesssteuerung vorgesehen ist. Anschließend werden einige Problemstellungen mittels eines etwas umfassenderen Szenarios dargestellt.

2.1 Modellierung und Ausführung von Geschäftsprozessen

PAIS bestehen aus einer Buildtime- und einer Runtime-Komponente. Zur Modellierungszeit (Buildtime) wird eine Prozessvorlage erstellt, die den später auszuführenden Geschäftsprozess beschreibt. Hierzu wird ein Prozessgraph modelliert, der aus Aktivitäten besteht, deren Ausführungsreihenfolge mittels Kanten und Bedingung festgelegt wird. Dieser Prozessgraph wird zur Ausführungszeit (Runtime) verwendet, um Prozessinstanzen zu erzeugen. Eine Prozess-Engine führt diese aus, indem für jede aktuell anstehende Aktivitäteninstanz (meist nur Aktivität genannt) entsprechende Einträge in den Arbeitslisten der potentiellen Bearbeiter erzeugt werden. Einer dieser Bearbeiter wählt einen solchen Eintrag aus, so dass er die Aktivität bearbeiten kann. Häufig besteht die Bearbeitung darin, ein Formular auszufüllen.

Der Kontrollfluss ist der am deutlichsten sichtbare Prozessaspekt. Er definiert die Ablaufreihenfolge mittels des Prozessgraphen (vgl. Abb. 1). Dessen Knoten stellen die Aktivitäten dar, die von Benutzern ausgeführt werden, automatisch durchgeführte Prozessschritte oder ganze Subprozesse repräsentieren. Außerdem enthält der Prozessgraph Gateways (Rauten), die Verzweigungen (Split) und Zusammenführung (Join) darstellen. In der Literatur findet sich eine Vielzahl an Pattern zur Modellierung des Kontrollflusses [RH06]. In kommerziellen PMS sind für Verzweigungen üblicherweise Split- und Join-Knoten mit XOR- (ein Zweig wird aufgrund einer Bedingung gewählt), OR- (mehrere Zweige) und AND-Semantik (alle Zweige) verfügbar. Außerdem werden Schleifen unterstützt. Sowohl Verzweigung wie auch Schleifen stellen eine einfache Form von vormodellierter Flexibilität dar, weil sie dazu führen, dass die Menge der ausgeführten Aktivitäten und deren Ausführungsreihenfolge bei jeder Prozessinstanz unterschiedlich sein können. Einige PMS ermöglichen zusätzlich die Instanziierung einer dynamisch festzulegenden Anzahl von identischen parallelen Zweigen, wobei deren Anzahl zum

Startzeitpunkt dieser „Multi-Instanz-Parallelität“ feststehen muss. Dies entspricht dem Kontrollfluss-Pattern „Multiple Instances with a priori Run-Time Knowledge” [RH06].

2.2 Herausforderungen bzgl. vormodellierbarer Flexibilität

Im Folgenden wird an einem Praxisbeispiel der Bedarf an Flexibilität aufgezeigt. Im Fokus hierbei liegen, wie bereits in Abschnitt 1 erläutert, nicht dynamische Änderungen mit denen auf völlig unerwartete Ereignisse reagiert wird. Stattdessen werden Situationen betrachtet, die zwar Ausnahmefälle darstellen, aber vorhersehbar sind. Deshalb kann bereits zur Buildtime ein geeignetes Verhalten vormodelliert werden.

Abb. 1 zeigt einen vereinfachten Change-Management-Prozess (CMP), wie er zur Beantragung von Produktänderungen in der Automobilindustrie dient. Die Notation ist an BPMN 2.0 angelehnt. Mit der Akt. A kann ein beliebiger Mitarbeiter des Automobilherstellers eine Änderung eines Bauteils (z.B. Form der Motorhaube) beantragen. Da die Ausführung einer CMP-Instanz einen erheblichen Aufwand verursacht, kann diese in Akt. B bzw. B' von einer Führungskraft gestoppt werden. Akt. C ermittelt automatisch den Verantwortlichen für das zu verändernde Bauteil durch eine Abfrage an das Produktdaten-Management-System (PDM-System). Dieser bewertet in Akt. D Aufwand und Nutzen der Änderung aus Sicht des Entwicklungsbereichs. In Akt. E identifiziert er betroffene Nachbarteile (z.B. Kotflügel, Kühler), die durch eine Formänderung der Motorhaube ebenfalls angepasst werden müssen. Akt. F erfragt die zugehörigen Bauteil-Details und -Verantwortlichen und speichert die Daten in der Liste Nachbarteile.

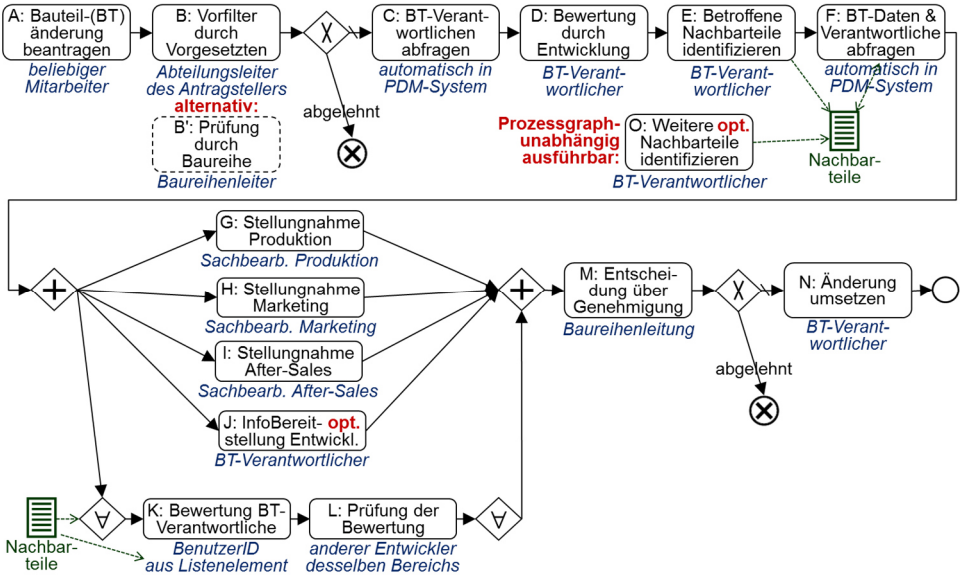

Abb. 1: Change-Management-Prozess (CMP) für Produktänderungen

In den Akt. G bis I bewerten diverse Bereiche zeitgleich (parallel), ob die Änderung aus ihrer Sicht realisierbar ist, und welche Kosten entstehen. Die Akt. J erlaubt es dem Bauteilverantwortlichen, hierzu zusätzliche Informationen zur Verfügung zu stellen (die z.B. telefonisch angefragt wurden). Die Bewertung der Änderung aus Sicht der Nachbarteile erfolgt in Akt. K durch den jeweiligen Bauteilverantwortlichen. Diese Aktivität wird mehrfach instanziiert (einmal je Nachbarteil). Dasselbe gilt für Akt. L, in der ein anderer Entwickler die Bewertung überprüft. In Akt. M wird über die Genehmigung des Änderungsantrags entschieden, und ggf. werden die Bauteile in Akt. N geändert.

Die Ausführung des CMP erfordert an einigen Stellen Flexibilität: So wurde die Akt. B' als Alternative zur Akt. B modelliert. B' wird verwendet, falls die Akt. B aktuell nicht ausführbar ist, z.B. weil der disziplinarische Vorgesetzte fachlich für diese Entscheidung nicht ausreichend kompetent ist. Die Akt. J wurde zur Buildtime als optional (opt.) eingestuft. Das bedeutet, dass sie entsprechend gekennzeichnet in der Arbeitsliste des Bauteilverantwortlichen erscheint. Dieser kann nun entscheiden, ob Bedarf an einer zusätzlichen Informationsbereitstellung besteht, oder ob er die Aktivität nicht ausführen möchte. Die Akt. K und L sind in eine Multi-Instanz-Parallelität eingebettet, d.h. die Anzahl Zweige ergibt sich beim ∀-Split aus der Länge der Liste Nachbarteile. Diese Liste wurde von den Akt. E und F befüllt. Sie kann aber auch noch nachträglich durch die Akt. O erweitert werden. Die Akt. O ist unabhängig von Prozessgraphen. Deshalb ist sie jederzeit ausführbar, macht aber nur bis zum Ende der Multi-Instanz-Parallelität Sinn (d.h. vor dem ∀-Join). Danach können zusätzliche Nachbarteile nicht mehr durch zusätzliche Instanzen der Akt. K und L berücksichtigt werden.

3 Konstrukte zur vormodellierten Flexibilität

Nachfolgend wird dargestellt, was genau zur Buildtime vormodelliert werden muss, um zur Runtime eine große Flexibilität zu ermöglichen, ohne dadurch viel Aufwand für den Benutzer zu verursachen. Hierbei wird ausschließlich der Aspekt Kontrollfluss (KF) betrachtet und die Notwendigkeit einiger Anforderungen wird mit Anwendungsbeispielen belegt. Bzgl. weiterer Prozessaspekte und für Anwendungsbeispiele für alle Anforderungen muss aus Platzgründen auf [Ba17] verwiesen werden.

KF-1: Optionale Aktivitäten

Bei optionalen Aktivitäten entscheidet der Benutzer, ob eine Aktivität überhaupt ausgeführt werden soll, oder ob sie bei dieser Prozessinstanz irrelevant ist. Hierzu wird die Aktivität in den Arbeitslisten angezeigt, so dass ein Benutzer sie starten kann. Der Arbeitslisteneintrag enthält jedoch zusätzlich eine Kennzeichnung, dass die Aktivität optional ist, und die Möglichkeit sie auszulassen (z.B. mittels eines entsprechenden Buttons bei diesem Eintrag). Bei einer optionalen Aktivität kann es sich auch um eine zusammengesetzte Aktivität handeln, so dass z.B. ein gesamter Subprozess übersprungen wird.

Zur Buildtime muss angegeben werden können (z.B. durch Setzen einer Markierung), dass

eine Aktivität optional sein soll (vgl. Akt. J in Abb. 1). Eine optionale Aktivität kann unter bestimmten Umständen auch hinfällig werden, ohne dass der Benutzer sich aktiv für ein „Auslassen“ entscheidet (s.u. KF-1b und c). Auch dies muss zur Buildtime konfigurierbar sein.

KF-1a: Der Normalfall bei optionalen Aktivitäten ist, dass diese ausgeführt werden. Um sie auszulassen, muss der Benutzer explizit aktiv werden. So wird beim CMP aus Abb. 1 die Akt. J dem Bauteilverantwortlichen so lange angeboten, bis er sie bearbeitet oder sich für das Auslassen entscheidet. Dies ist sinnvoll, weil die erzeugte Information sowohl von den parallel ausgeführten Akt. G bis I, wie auch von der Nachfolgeakt. M genutzt werden kann.

KF-1b: Mit der optionalen Akt. C in Abb. 2 kann die ursprünglich in Akt. A erstellte Bauteilbeschreibung detailliert werden, falls sie sich durch die Ausführung von Akt. B signifikant verändert hat. Diese Bauteilbeschreibung wird von der Akt. E in das Stücklistensystem übertragen. Allerdings wartet Akt. E nicht auf die Ausführung oder das Auslassen von Akt. C, weil die Stücklistendaten von anderen Prozessen dringend benötigt werden und Verzögerungen nicht akzeptabel wären. Ist jedoch der untere Zweig beendet (d.h. der AND-Join erreicht), so macht die Ausführung von Akt. C keinen Sinn mehr. Deshalb wird sie von der Prozess-Engine dann automatisch ausgelassen (sofern dies zur Buildtime so festgelegt wurde).

KF-1c: Das Beispiel aus Abb. 2 kann auch so abgewandelt werden, dass ein anderes Verhalten erforderlich wird: Befinden sich an der mit * markierten Stelle (d.h. nach Akt. E) weitere Aktivitäten, so ist Akt. C nicht erst dann hinfällig, wenn der untere Zweig den AND-Join erreicht. Stattdessen muss modelliert werden, dass Akt. C automatisch ausgelassen wird, sobald die Akt. E ausführt wurde. Akt. E stellt somit eine Art Milestone dar.

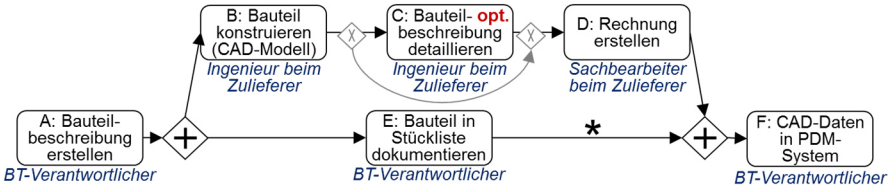

Abb. 2: Konstruktion eines Bauteils durch einen Zulieferer

Optionale Aktivitäten lassen sich nicht einfach mittels eines normalen XOR realisieren (grau in Abb. 2 angedeutet). Abgesehen davon, dass sich KF-1b und c so nicht umsetzen lassen, ergibt sich auch eine andere (bzw. keine) Benutzerinteraktion: Bereits am XOR-Split wird die Verzweigungsentscheidung anhand einer Regel und mittels Werten von Prozessvariablen getroffen. Im Falle einer Auslassung würde die Akt. C also erst gar nicht in die Arbeitslisten der potentiellen Bearbeiter eingestellt werden. Im anderen Fall wäre sie nicht mehr auslassbar. Dies entspricht nicht dem bei optionalen Aktivitäten gewünschten Verhalten, so dass es sich hierbei tatsächlich um ein eigenständiges Kontrollfluss-Konstrukt handelt. Aus ähnlichen Gründen (und weil dies zu einem sehr unübersichtlichen Prozessgraph führen würde) lassen sich auch die im Folgenden dargestellten

Anforderungen nicht mit (normalen) XOR-Gateways realisieren (z.B. KF-2 und KF-3).

KF-2: Alternative Aktivitäten

Die Notwendigkeit für alternative Aktivitäten soll an einem medizinischen Beispiel motiviert werden: Ein Patient erhält normalerweise einen Wirkstoff X in Form von Tabletten (Akt. A). In Sonderfällen erkennt der Arzt, dass der Wirkstoff X bzw. die orale Einnahme ungeeignet ist. Deshalb entscheidet er sich für die alternative Akt. B und verabreicht den Wirkstoff Y per Spritze.

Das PMS verhält sich bei alternativen Aktivitäten folgendermaßen: Die Standardaktivität A wird den Benutzern in ihren Arbeitslisten angezeigt. Zusätzlich dargestellt wird ein Hinweis, dass es (eine) alternative Aktivität(en) gibt. Der Benutzer kann sich durch eine aktive Aktion in der Arbeitsliste für eine alternative Aktivität entscheiden (außer bei KF-2d). Zur Buildtime wird festgelegt, welche der folgenden Varianten verwendet werden soll, d.h. wann und wie die Entscheidung für eine alternative Aktivität getroffen wird.

KF-2a: Der Benutzer entscheidet dies vor Reservierung (d.h. dem Start) der Aktivität. Dies ist der Normalfall (und auch der einfachste Fall).

KF-2b: Der Benutzer kann sich auch noch während der laufenden Bearbeitung von Akt. A für Akt. B entscheiden. Dann wird die reguläre Akt. A automatisch abgebrochen und Akt. B gestartet.

KF-2c: Der Benutzer erkennt viel später im Prozess, dass die alternative Aktivität geeigneter gewesen wäre. Die alternative Aktivität wird dann später und zusätzlich ausgeführt. Dies macht z.B. bei Aktivitäten zur Datenerfassung Sinn, wenn durch die alternative Aktivität mehr bzw. andere Daten erfasst werden, die sich im Kontext des aktuellen Geschäftsvorfalles (verspätet) als erforderlich herausgestellt haben. Die zuvor erzeugten Prozessdaten werden also nachträglich verändert.

KF-2d: Die alternative Aktivität wird von der Prozess-Engine automatisch ausgewählt, falls die eigentlich vorgesehene Aktivität fehlschlägt. Hierbei kann es sich um einen fehlgeschlagenen Service-Aufruf bei automatischen Aktivitäten handeln. Aber auch bei manuell bearbeiteten Aktivitäten kann die Prüfung der Nachbedingungen ergeben, dass sie nicht korrekt ausgeführt wurden (z.B. fehlende oder inkonsistente Ergebnisdaten).

Zur Buildtime muss auch festgelegt werden, wer das Recht hat, sich für die Alternative zu entscheiden. Meist wird ein (potentieller) Bearbeiter dieser Aktivität die Entscheidung treffen dürfen. Es soll aber auch möglich sein, diesen Personenkreis einzuschränken, z.B. auf (besonders kompetente) Personen, die eine zusätzliche Rolle innehaben.

Falls die alternative Aktivität zusammengesetzt ist, entscheidet man sich für die Ausführung eines anderen Subprozesses, anstatt einer elementaren Aktivität. Dann ist nicht nur der nächste Aktivitätenbearbeiter von dieser Entscheidung betroffen, sondern auch alle anderen Bearbeiter des Subprozesses. Deshalb muss die Entscheidung evtl. durch einen speziellen Verantwortlichen (z.B. Projektleiter) getroffen werden, der selbst nicht

zwangsweise ein Bearbeiter von Aktivitäten des Subprozesses ist.

Selbstverständlich können für eine Aktivität auch mehrere alternative Aktivitäten vormodelliert werden, aus denen der Benutzer dann eine geeignete auswählen kann.

KF-3: Sprünge innerhalb des Prozessgraphen

KF-3a: Sprünge vorwärts Angenommen, bei einem Reiseantragsprozess erfolge die Genehmigung nach diversen Stellungnahmen und Kostenschätzungen, die lange dauern können. Bei Reiseanträgen für kurzfristige Termine kann es vorkommen, dass ausnahmsweise, ausgehend von diesen Aktivitäten, direkt zur Genehmigung gesprungen werden muss, weil sonst ein extrem wichtiger Termin verpasst wird und wirtschaftlicher Schaden entsteht.

Im einfachsten Fall findet ein Sprung statt, bevor die Quellaktivität (d.h. Startpunkt des Sprungs) gestartet wurde. Falls die Quellaktivität bereits gestartet wurde, muss sie vor dem Sprung abgebrochen werden. Bei der Modellierung muss festgelegt werden, ob ausgehend von dieser Quellaktivität ein Sprung erlaubt ist, ob sie hierfür (automatisch) abgebrochen werden darf und welches mögliche Zielaktivitäten des Sprungs sind. Außerdem muss festgelegt werden, wer einen solchen Sprung auslösen darf, z.B. der aktuelle Bearbeiter der Aktivität oder ein fachlicher Prozessverantwortlicher.

Für die übersprungenen Aktivitäten ist festzulegen, ob sie nachgeholt werden sollen. Der Normalfall ist, sie endgültig auszulassen. Es kann jedoch auch erforderlich sein, dass eine übersprungene Aktivität nachgeholt wird. Dies ist zur Buildtime festzulegen, inkl. einer Angabe, ob das Nachholen zu einem beliebigen späteren Zeitpunkt erfolgen darf, oder ob es spätestens vor dem Start einer bestimmten anderen Aktivität erfolgen muss.

Eine Modellierung der Sprünge mittels Kanten und (normalen) XOR-Splits ist kaum sinnvoll. Wie bereits bei KF-1 diskutiert, werden die Bedingungen des Split-Knotens zu früh ausgewertet. Hinzu kommt, dass zahlreiche XOR-Splits und entsprechende Kanten eingefügt werden müssen (z.B. im Reiseprozess ausgehend von allen Bewertungen), wodurch der Prozessgraph sehr unübersichtlich wird. Deshalb solle ein Modellierungswerkzeug die Sprungkanten ausblenden und vom regulären Kontrollfluss unterscheidbar visualisieren können. Außerdem sollte die Festlegung mehrerer Quell-/Zielaktivitäten als gemeinsame Region möglich sein.

KF-3b: Sprünge rückwärts Angenommen, beim CMP aus Abb. 1 wird während der Ausführung von Akt. E bis L entdeckt, dass bei Akt. D fehlerhafte Daten eingegeben wurden. Deshalb soll keine Genehmigung (Akt. M) erfolgen, sondern zur Akt. D zurückgesprungen und diese wiederholt werden. Danach wird der Prozess (mit korrekten Daten) ab Aktivität E erneut durchlaufen. Hier sind mehrere Quellknoten (Akt. E bis L) für den Sprung vorgesehen. Ebenso wie bei Vorwärtssprüngen (KF-3a) sollen diese nicht einzeln mit Kanten und XOR-Splits modelliert werden, sondern mittels Prozessregionen. Außerdem muss festgelegt werden, wer das Recht hat, einen solchen Sprung auszulösen.

Da nach einem Rücksprung der Prozessgraph wieder vorwärts durchlaufen wird, muss für

jede Aktivität festgelegt werden, was dann mit ihren ursprünglichen Ergebnissen (Output-Daten) passieren soll. Hier sind 3 Varianten denkbar und es hängt von der „inhaltlichen Bedeutung“ der Aktivität ab, welche geeignet ist.

1. Verwerfen: Die ursprünglichen Ergebnisse werden verworfen und die Aktivität wird „ganz normal“ erneut ausgeführt
2. Kontrollieren: Die Aktivität wird zwar nochmals ausgeführt, die ursprünglichen Output-Daten bleiben aber erhalten. Dem Benutzer wird ein vorausgefülltes Formular angezeigt und er kann diese Daten nun kontrollieren und ggf. korrigieren.
3. Beibehalten: Die Aktivität wird nicht erneut ausgeführt und ihre Output-Daten bleiben unverändert erhalten.

Es wird nun am Beispiel des CMP erläutert, warum alle 3 Varianten erforderlich sind: Zum Zeitpunkt des oben erwähnten Rücksprungs von Akt. L zu D seien die Akt. G bis I bereits beendet. Für die Akt. H wurde „Beibehalten“ (3.) festgelegt. Diese Aktivität muss nach dem Rücksprung nicht nochmals ausgeführt werden, weil durch Akt. D geänderte Entwicklungsdetails für das Marketing irrelevant sind. Für Akt. G wird „Kontrollieren“ (2.) festgelegt. Die Ergebnisse dieser Aktivität werden sich selten verändern, müssen jedoch geprüft und ggf. angepasst werden. Eine solche Veränderung ergibt sich aus Sicht der Fahrzeugproduktion, falls durch die Bauteiländerung die Montage schwieriger geworden ist. Für Akt. I wird „Verwerfen“ (1.) verwendet, d.h. die Aktivität muss vollständig neu ausgeführt werden. Ihre ursprünglichen Output-Daten sind irrelevant, weil ein geändertes Bauteil im Reparaturfall andere Beschaffungs- und Einbaukosten verursacht. Die Varianten 1 und 2 (sofern anwendbar) haben den Vorteil, dass beim erneuten Durchlaufen der Aktivitäten Zeit und Arbeitsaufwand eingespart werden.

In manchen Fällen ist es bei einem Rücksprung erforderlich, dass die ursprünglich ausgeführten Aktivitäten rückgängig gemacht werden. Wird z.B. bei einem Bestellprozess über die Auftragsvergabe an einen Zulieferer zurückgesprungen, so muss diese widerrufen werden oder der Lieferant muss informiert werden, dass er diesen Auftrag ruhen lassen soll. Hierfür muss eine Kompensationsaktivität modelliert werden, die genau diese Aufgabe übernimmt. Hierbei handelt es sich im Prinzip um eine normale Aktivität, die aber speziell für Rücksprungaktionen modelliert wird (also nicht zum regulären Kontrollfluss gehört).

KF-3c: Sprünge bei Parallelitäten Bei Sprüngen in einen parallelen Bereich hinein oder aus einem heraus ergeben sich einige zusätzliche Fragestellungen. Die möglichen Quell- und Zielregionen werden zur Buildtime festlegt (ebenso wie die bereits beschriebenen Berechtigungen etc.). Aufgrund der Parallelität wird hierfür jeweils eine Menge von Aktivitäten (eine Region) angegeben. Möchte ein Benutzer nun einen Sprung auslösen, so muss er eine Aktivität je parallelem Zweig als Sprungziel festlegen.

Vorwärtssprung: Um den Aufwand für den Benutzer zu reduzieren, kann für jeden Zweig ein Default-Zielknoten festgelegt werden, z.B. die erste Aktivität nach dem AND-Split. Dann kann der Benutzer einen Sprung auslösen, bei dem er nur für einige der Zweige eine Zielaktivität angibt. Bezüglich dem Nachholen der übersprungenen Aktivitäten gibt

es wieder die in KF-3a beschriebenen Varianten.

Rückwärtssprung: Zur Buildtime ist für jeden Zweig festzulegen, wie sich dessen übersprungene Aktivitäten verhalten sollen. Hierfür gibt es folgende Möglichkeiten:

- Eine aktuell laufende Aktivität des Zweiges wird abgebrochen. So soll z.B. eine laufende Aktivität abgebrochen werden, deren Ergebnisse bei der erneuten Ausführung ohnehin verworfen werden (s. KF-3b: Verwerfen). Diese Aktivität fertig zu bearbeiten oder sogar neu zu starten wäre verschwendeter Aufwand.
- Eine aktuell laufende Aktivität in dem Zweig kann zu Ende ausgeführt werden. Es werden aber keine Nachfolgeraktivitäten gestartet. So kann z.B. bei KF-3b (Kontrollieren) eine bereits gestartete Aktivität beendet werden, damit der bisher investierte Arbeitsaufwand nicht verloren geht. Eine Ausführung von Nachfolgeraktivitäten ist jedoch unerwünscht, weil bei KF-3b deren Ergebnisse evtl. verworfen werden.
- Der parallele Zweig kann bis zum AND-Join weiter bearbeitet werden. Dies kann sinnvoll sein, wenn die Ergebnisdaten ohnehin verwendbar sind (s. KF-3b: Beibehalten und evtl. Kontrollieren). Dann steht für die Bearbeitung der Aktivitäten dieses Zweiges viel Zeit zur Verfügung, bis die Ausführung nach dem Rücksprung wieder beim AND-Join ankommt. Dadurch sinkt die Gefahr, dass beim erneuten Durchlaufen Verzögerungen entstehen, weil auf die Beendigung dieser Aktivitäten gewartet werden muss.

Für die Zielknoten des Sprungs können zur Buildtime wieder Defaults angegeben werden. Dies kann jede Aktivität eines Zweiges sein. Häufig wird dies die erste Aktivität eines Zweiges sein, so dass der gesamte Zweig wiederholt wird. Es muss aber auch festgelegt werden können, dass ein Zweig gar nicht erneut ausgeführt werden soll. Für die wiederholt auszuführenden Aktivitäten gibt es wieder die KF-3b dargestellten Varianten.

KF-4: Multi-Instanz-Parallelität

Ein Beispiel hierfür wurde bereits in Abschnitt 2.2 vorgestellt. So soll im CMP aus Abb. 1 ein Bauteil verändert werden. Da sich dadurch evtl. seine Form, Material oder Härte ändert, kann dies benachbarte Bauteile beeinträchtigen. Eine Liste (und damit die Anzahl) dieser Nachbarteile wird in Akt. E festgelegt. Entsprechend oft müssen die Akt. K und L ausgeführt werden, jeweils mit unterschiedlichen Input-Daten und Bearbeitern.

KF-4a: Im (bisher erläuterten) einfachsten Fall ist zur Runtime beim Beginn der Multi-Instanz-Parallelität (d.h. beim ∀-Split) bekannt, wie viele parallele Zweige erzeugt werden müssen (Instanzen der Akt. K und L). Dies entspricht dem Kontrollfluss-Pattern „Multiple Instances with a priori Run-Time Knowledge“ [RH06].

KF-4b: Durch Ausführung der Akt. O können nachträglich weitere Nachbarteile hinzukommen, so dass zusätzliche parallele Zweige erzeugt werden müssen. Für diese werden also später Instanzen der Akt. K erzeugt. Dies entspricht der Variante „without a priori

Run-Time Knowledge“ [RH06]. Weitere parallele Zweige können hierbei so lange entstehen, bis alle existierenden Zweige der Multi-Instanz-Parallelität abgeschlossen sind.

KF-4c: Als Erweiterung hiervon kann es auch sinnvoll sein, zur Buildtime ein anderes Kriterium dafür festzulegen, wie lange weitere Zweige erzeugt werden dürfen. Eine mögliche Regel hierfür ist, dass nur solange neue Zweige hinzukommen dürfen, bis ein Milestone in einem der Multi-Instanz-Zweige erreicht ist. So sind z.B. keine neuen Zweige mehr möglich, wenn die erste Bewertung durch einen Bauteilverantwortlichen (Akt. K) abgeschlossen ist. Diese wurde zwar noch nicht von einem Kollegen geprüft (Akt. L), so dass der Zweig nicht vollständig beendet ist. Jedoch hat sich der Bearbeiter von Akt. K auf die ursprüngliche Bauteilliste verlassen, so dass diese nicht mehr nachträglich verändert werden darf. Ein anderer Typ von Regel verwendet Milestones in einem parallelen Zweig. Wenn z.B. in Akt. G die Produktion basierend auf der aktuellen Teileliste ihre Stellungnahme abgegeben hat, dann darf diese Liste nicht mehr verändert werden, so dass auch keine parallelen Zweige mehr erzeugt werden können.

KF-5: Start-Ende-Abhängigkeiten zwischen Aktivitäten

Die in [RH06] vorgestellten Kontrollfluss-Pattern berücksichtigen ausschließlich die Reihenfolge vollständig ausgeführter Aktivitäten, z.B. Akt. A muss beendet sein bevor B gestartet werden kann. Dies lässt sich erweitern, indem der Start und das Ende von Aktivitäten getrennt betrachtet werden. Dadurch werden zusätzliche (explizit erlaubte) Ausführungsreihenfolgen möglich und damit wird die Flexibilität erhöht. Hierfür gibt es 4 Möglichkeiten, wobei die erste einer normalen Sequenz entspricht:

KF-5a: EndBeforeStart Ende von Akt. A muss vor Start von Akt. B erfolgen

KF-5b: StartBeforeStart Start von Akt. A muss vor Start von Akt. B erfolgen

KF-5c: EndBeforeEnd Ende von Akt. A muss vor Ende von Akt. B erfolgen

KF-5d: StartBeforeEnd Start von Akt. A muss vor Ende von Akt. B erfolgen

Beispielsweise ist der Typ EndBeforeEnd (KF-5c) in folgendem Szenario sinnvoll: Die Akt. A (Fahrzeug reinigen) muss beendet werden, bevor die Akt. B (Fahrzeug an Kunde ausliefern) abgeschlossen wird. Es ist nicht möglich, das Fahrzeug danach noch zu reinigen. Es ist jedoch erlaubt, dass die Akt. A und B überlappend ausgeführt werden, wenn z.B. das Fahrzeug während einer Transportpause gereinigt wird.

KF-5e: Optionale Abhängigkeiten Als Erweiterung sollen zudem Start-Ende-Abhängigkeiten verwendbar sein, die eine optionale Ordnung festlegen. Eine solche Reihenfolge soll eingehalten werden, muss aber nicht. Die idealerweise als erstes auszuführende Aktivität wird in der Arbeitsliste entsprechend gekennzeichnet, aber auch die optionalen Nachfolger sind dort sichtbar. Der Benutzer kann sich also entscheiden, eine dieser Aktivitäten zuerst zu bearbeiten.

KF-5f: Zeitabstände Zusätzlich zur Reihenfolgebeziehung kann es erforderlich sein, dass zwischen den Aktivitäten zeitliche Minimal- und Maximalabstände existieren. So muss z.B. ein in Akt. A gehärtetes Bauteil mind. 24 Stunden abkühlen, bevor es lackiert werden

darf (Akt. B). Die Akt. B erscheint deswegen erst dann in den Arbeitslisten der Benutzer, wenn diese 24 Stunden verstrichen sind. Solche zeitlichen Abstände werden ebenfalls im Prozessmodell definiert, sind durch die Prozess-Engine sicherzustellen und können sich auf die Start- und Endezeitpunkte der Aktivitäten beziehen.

KF-5g (Mutual-Exclusion): Von den in ein solches Konstrukt eingebetteten Aktivitäten darf stets nur eine einzige bearbeitet werden, die anderen müssen vollständig davor oder vollständig danach bearbeitet werden.

KF-6 Prozessgraph-unabhängige Aktivitäten

Es kann Aktivitäten in einem Geschäftsprozess geben, die den Benutzern zusätzlich angeboten werden sollen, d.h. die nicht in den Prozessablauf eingebunden sind. Diese können spontan, aufgrund der Entscheidung eines Benutzers ausgeführt werden. Im CMP aus Abb. 1 kann jederzeit festgestellt werden, dass in Akt. E ein Nachbarteil vergessen wurde, so dass die Prozessgraph-unabhängige Akt. O ausgeführt wird. Diese ergänzt die Liste der Nachbarteile, so dass ein zusätzlicher Zweig der Multi-Instanz-Parallelität gestartet wird (ggf. nachträglich, vgl. KF-4). Eine solche Prozessgraph-unabhängige Aktivität kann vom Benutzer z.B. über das Programm-Menü gestartet werden.

Zur Buildtime müssen Prozessgraph-unabhängige Aktivitäten (wie normale Aktivitäten) vollständig modelliert werden, inkl. einer Bearbeiterzuordnung, der Anbindung ihrer Input-/Output-Parameter an Prozessvariablen (hier: Liste Nachbarteile), etc. Zur Runtime können sie weitgehend wie normale Aktivitäten verwendet werden, so dass für die Benutzer ein viel geringerer Aufwand entsteht (vgl. Abschnitt 1), als für das dynamische Einfügen [RD98] einer zusätzlichen Aktivität.

Für Prozessgraph-unabhängige Aktivitäten muss zudem festgelegt werden, ob sie mehrfach ausgeführt werden dürfen. Ggf. ist zusätzlich eine „Prozessregion“ erforderlich, die angibt, wann diese Aktivität ausführbar ist. Beim CMP darf die Akt. O zwar beliebig oft, aber nur vor Ende der Multi-Instanz-Parallelität ausgeführt werden (vor dem ∀-Join).

KF-7 Benutzer-Aktionen

Ein Benutzer kann spontane Aktionen durchführen, die den Ausführungszustand einer Prozessinstanz, und damit den Kontrollfluss-Ablauf, verändern. Im Gegensatz z.B. zu optionalen Aktivitäten (KF-1) ist eine solche Aktion selbst nicht vorgeplant, d.h. im Prozessmodell nicht enthalten. Dem Benutzer wird auch nichts Spezielles in seiner Arbeitsliste dargestellt. Stattdessen werden solche Aktionen mittels des Programmmenüs oder stets vorhandener Buttons ausgelöst. Für die nachfolgend erwähnten Aktionen sollte für den betroffenen Prozesstyp bzw. die Aktivität zur Buildtime zumindest festgelegt werden, ob sie überhaupt erlaubt sind und wer die Rechte hierfür besitzt.

KF-7a: Abbruch einer Prozessinstanz

KF-7b: Abbruch einer Aktivität

KF-7c: Überspringen einer Aktivität (Skip)

KF-7d: Kompensation einer Aktivität (Undo)

KF-7e: Wiederholung einer Aktivität (Redo).

Für KF-7a, d und e kann zusätzlich definiert werden, in welchem Prozess-Ausführungszustand die Aktion erlaubt ist. So ist ein Undo einer Aktivität auch denkbar, lange nachdem die ursprüngliche Bearbeitung stattgefunden hat. Für ein Undo kann zudem eine Kompensationsaktivität modelliert werden (ansonsten werden nur die Prozessinstanz-Daten zurückgesetzt) und nach dem Undo kann auch noch ein Redo erlaubt bzw. erzwungen werden (für Details und Praxisbeispiele siehe aus Platzgründen [Ba17]).

4 Diskussion

In [KN06] werden verschiedene Arten von Flexibilität für Geschäftsprozesse vorgestellt. Die Kategorien werden in [Sc07] noch verfeinert, so dass die für uns relevanten Kategorien „Flexibility by Design“ und „Flexibility by Underspecification“ entstehen.

In der Literatur finden sich kaum Arbeiten, die die Fragestellung behandeln, was zur Buildtime vormodelliert werden muss, um zur Runtime eine große Flexibilität bei gleichzeitig geringem Aufwand für die Benutzer zu erreichen. Der Ansatz, der in der Literatur am häufigsten zur Vormodellierung von Sonderfällen und Ausnahmebehandlungen vorgeschlagen wird, ist ein Exception-Handling auf Basis von Events und Exception-Handlern [Le10, RW12]. Für Aktivitäten oder ganze Regionen wird ein (ggf. unterbrechendes) Event definiert. Falls dieses zur Runtime auftritt (Throw), führt es zur Ausführung eines Exception-Handlers (Catch). Dieses Vorgehen ähnelt dem in Programmiersprachen (Try-Catch-Block) und eignet sich insb. zum Abfangen von technischen Fehlern wie z.B. einem Abbruch des Aktivitätenprogramms. Es ist für einige der vorgestellten Anforderungen überhaupt nicht geeignet (z.B. optionale Aktivitäten, Start-Ende-Abhängigkeiten), weil hier gar kein Ausnahmefall vorliegt.

Komplexe Kontrollfluss-Pattern ermöglichen eine gewisse Art von Flexibilität [RH06 RW12]. Allerdings liegt hier der Fokus nicht auf Anforderungen bzgl. der Vormodellierung von Sachverhalten, mit dem Ziel eine große Flexibilität zur Runtime zu erhalten. Dasselbe gilt für weitere Arbeiten zur Flexibilisierung von Prozessen, wie z.B. Schemaevolution [Ri04], Variantenmanagement [RHB15], Constraint-basierte Prozesse [Bo11, Pe07, RW12], etc. Für eine ausführliche Diskussion dieser Arbeiten muss aus Platzgründen auf [Ba17] verwiesen werden.

5 Zusammenfassung und Ausblick

In PMS muss vom vorgegebenen Prozess abgewichen werden können, damit diese Systeme in der Praxis benutzbar sind. Dynamische Änderungen sind bei vorhersehbaren Abweichungen jedoch ein zu großer Aufwand und eine unnötige Fehlerquelle. In diesem Beitrag werden Anforderungen und zugehörige Praxisbeispiele vorgestellt, die zeigen, wie

vorhersehbare Sonderfälle und Ausnahmebehandlungen bereits zur Buildtime vormodelliert werden können. Dies soll auch einen Beitrag dazu leisten, Tool-Hersteller zur Unterstützung der beschriebenen Szenarien in kommerziellen PMS zu motivieren.

Die Relevanz und Vollständigkeit der Szenarien sollte noch anhand von weiteren Praxisbeispielen verifiziert werden. Dies wird dadurch erschwert, dass einige der vorgestellten Konstrukte bei einer klassischen Prozessmodellierung nicht vorgesehen sind (z.B. Start-Ende-Abhängigkeiten existieren nicht bei EPKs und BPMN). Deshalb werden entsprechende Situationen in bereits existierenden Prozessmodellen evtl. nicht erfasst worden sein, obwohl sie eigentlich existieren. Einzelne der vorgestellten Anforderungen sollen im Projekt CoPMoF noch weiter detailliert werden. Dies ist z.B. für Sprünge (KF-3) erforderlich. Hier muss die gewünschte Ausführungssemantik noch formal und damit eindeutig definiert werden, weil diese im Zusammenspiel mit Parallelitäten nicht offensichtlich ist. Dasselbe gilt für Start-Ende-Abhängigkeiten (KF-5).

6 Literaturverzeichnis

[Ba17] Bauer, T.: Vormodellierte Flexibilität in Prozess-Management-Systemen: Anforderungen, Vorgehensweisen, Lösungsansätze. Technischer Bericht (HNU WP 37), 2017.

[Bo11] Boukhebouze, M. et al.: A Rule-based Approach to Model and Verify Flexible Business Processes. In Int. Journal of Business Process Integration and Management, 2011, 5; S. 287–307.

[KN06] Kumar, K.; Narasipuram, M.: Defining Requirements for Business Process Flexibility. In Workshop on Business Process Modeling, Design and Support, Proc. of CAiSE06 Workshops, 2006; S. 137–148.

[Le10] Lerner, B. S. et al.: Exception Handling Patterns for Process Modeling. In IEEE Transactions on Software Engineering, 2010, 36; S. 162–183.

[Pe07] Pesic, M. et al.: Constraint-based Workflow Models: Change Made Easy. In Proc. 15th Int. Conf. on Cooperative Information Systems, 2007; S. 77–94.

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

[Re09] Redding, G. et al.: Modelling Flexible Processes with Business Objects. In Proc. IEEE Conf. on Commerce and Enterprise Computing, 2009; S. 41–48.

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

[RHB15] Reichert, M.; Hallerbach, A.; Bauer, T.: Lifecycle Management of Business Process Variants. In (J. vom Brocke, M. Rosemann Hrsg.): Handbook on Business Process Management, 2nd Edition. Springer, 2015; S. 251–278.

[Ri04] Rinderle, S.: Schema Evolution in Process Management Systems. Dissertation, Universität Ulm, 2004.

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

[Sc07] Schonenberg, M. et al.: Towards a Taxonomy of Process Flexibility (Extended Version), Eindhoven University of Technology, 2007.