Informatik Forsch. Entw. (2002) 17: 177–197
Digital Object Identifier (DOI) 10.1007/s00450-002-0122-0

Informatik
Forschung und Entwicklung

Effiziente Verträglichkeitsprüfung und automatische Migration von Workflow-Instanzen bei der Evolution von Workflow-Schemata*

Stefanie Rinderle, Manfred Reichert, Peter Dadam

Abteilung Datenbanken und Informationssysteme, Universität Ulm, 89069 Ulm (e-mail: {rinderle, reichert, dadam}@informatik.uni-ulm.de)

Eingegangen am 2. Juli 2002 / Angenommen am 15. Oktober 2002

Zusammenfassung. Sollen Workflow-Management-Systeme (WfMS) in umfassender Weise für die rechnerbasierte Verwaltung und Steuerung von Geschäftsprozessen einsetzbar sein, müssen die von ihnen verwalteten Workflow-Schemata und -Instanzen bei Bedarf rasch anpassbar sein. Dabei müssen die auf Basis eines (alten) Workflow-Schemas erzeugten Instanzen auch nach dessen Änderung ungestört weiterlaufen können, etwa durch Bereitstellung geeigneter Versionskonzepte. Sehr viel schwieriger wird es, wenn die angewandten Schemaänderungen – wo gewünscht und möglich – auch auf die bereits (vielleicht in großer Zahl) laufenden Workflow-Instanzen übertragen werden sollen. Dies bei Bedarf zu können – und zwar ohne Inkonsistenzen oder Fehler zu verursachen – ist aber ungemein wichtig, wenn ein WfMS breit und flexibel einsetzbar sein soll. In diesem Beitrag wird ein Ansatz zur effizienten Prüfung der Verträglichkeit von Workflow-Instanzen mit einem geänderten Workflow-Schema vorgestellt. Durch Einbeziehung aller Beschreibungskonstrukte (z.B. auch Schleifen und Datenflüsse) und damit zusammenhängender Fragestellungen wird darüber hinaus zum ersten Mal die Grundlage für ein umfassendes Änderungsmanagement geschaffen. Außerdem wird aufgezeigt, wie der Benutzer bei der Migration verträglicher Instanzen auf das neue Schema konkret unterstützt werden kann.

Schlüsselwörter: Workflow-Management – Adaptive Workflows – Schema-Evolution – Verträglichkeit – Migration

Abstract. A promising technology to enhance the flexibility of business processes is offered by workflow management systems (WfMS). In principle, changes of the process logic of application systems can be easily accomplished by modifying the (graphical) workflow (WF) schema accordingly. In doing so, it is extremely important that already running WF instances will not be disturbed. In current WfMS, this is achieved by the use of simple versioning concepts. WF schema adaptations, however, get much more difficult if the applied changes are to be propagated to already running WF instances as well. On the one hand such a feature is indispensable for many process-centered applications, on the other hand dynamic changes must not violate WF consistency. This paper presents a comprehensive framework for propagating WF schema changes to in-progress WF instances. In particular, we show how this can be done in an efficient manner and without causing inconsistencies or errors in the sequel. We establish well-defined criteria for checking whether a particular WF instance is compliant with the new WF schema or not, and we indicate which information becomes necessary in this context. Furthermore, we discuss how compliant WF instances can be automatically migrated to the new schema.

Keywords: Workflow management, Adaptive workflows, Schema evolution, Compliance, Migration

CR Subject Classification: H.4, J.1

1 Einleitung

Für Unternehmen gewinnt die elektronische Unterstützung ihrer Geschäftsprozesse zunehmend an Bedeutung. Sowohl für traditionelle Anwendungssysteme als auch für Anwendungen im sich rasch entwickelnden E-Business-Bereich wird in verstärktem Maße eine aktive Prozessunterstützung gewünscht [8,12,17]. Prozessorientierte Informationssysteme sollen die Durchführung unternehmensweiter und -übergreifender Abläufe aktiv koordinieren, Anwendungskomponenten prozessorientiert integrieren, Benutzer ablaufbezogen unterstützen, den Fortgang der Prozesse überwachen und ihren realen Verlauf möglichst lückenlos dokumentieren [20,28].

Sollen Geschäftsprozesse in solch umfassender Weise unterstützt werden, ist zu beachten, dass prozessorientierte Anwendungen sehr viel häufiger angepasst werden müssen als funktionsorientierte Informationssysteme [12,23]. Änderungen können etwa erforderlich werden, wenn neue gesetzliche Bestimmungen in Kraft treten, optimierte oder neu gestaltete Prozesse umgesetzt werden sollen [18] oder auf ein verändertes Marktgeschehen reagiert werden muss [17]. Wichtig ist, dass notwendige Prozessänderungen bei Bedarf

* Diese Arbeit wurde im Rahmen des Projekts „Änderungsmanagement in adaptiven Workflow-Management-Systemen" der Deutschen Forschungsgemeinschaft (DFG) erstellt.

rasch erfolgen können. Idealerweise sollte auch die Möglichkeit bestehen, die bereits laufenden „alten" Prozesse – soweit gewünscht und sinnvoll – auf die neue Abwicklung umzustellen. Eine solche Prozessunterstützung findet man bei heutigen betrieblichen Informationssystemen entweder gar nicht oder aber sie ist höchst unflexibel implementiert. Häufig ist die Ablaufsteuerung direkt in den Anwendungsprogrammen codiert, was die Anwendungsentwicklung sehr aufwendig und fehlerträchtig gestaltet.

Eine vielversprechende Technologie bieten *Workflow-Management-Systeme* (WfMS) [20,25,28]. Charakteristisch ist hier die Trennung von Prozesslogik und Anwendungscode, d.h. die Ablauflogik der unterstützten Arbeitsprozesse (engl. *Workflow*, WF) wird dem WfMS explizit durch (graphische) Modellierung der Prozesse bekannt gemacht und nicht im Programmcode „versteckt". Zu diesem Zweck muss für jeden zu unterstützenden *Prozess-Typ* ein *WF-Schema* erstellt und im System hinterlegt werden. Ein solches WF-Schema beschreibt die verschiedenen Aspekte eines Arbeitsprozesses wie Kontroll- und Datenflüsse, Bearbeiterzuordnungen oder Ausnahmebehandlungen. Des Weiteren können den *WF-Schritten* (*Aktivitäten*) Anwendungskomponenten zugeordnet werden, die dann während der WF-Bearbeitung zur Ausführung kommen. Ausgehend von einem solchen WF-Schema können neue *WF-Instanzen* erzeugt werden, für die das WfMS die Durchführung von Aktivitäten koordiniert, anstehende Aktivitäten den Anwendern über *Arbeitslisten* anbietet, deren fristgerechte Durchführung überwacht und die zugehörigen Anwendungsprogramme (mit den richtigen Daten) aufruft.

1.1 Problembeschreibung

Anpassungen können einen WF-Typ (bzw. sein Schema) als Ganzes oder auch nur einzelne Instanzen betreffen. Bei Änderungen auf Typebene ist zu fordern, dass die auf Basis des alten WF-Schemas erzeugten Instanzen ungestört weiterlaufen. Dies lässt sich z.B. durch geeignete Versionierungskonzepte für WF-Schemata erreichen, gemäß denen zukünftige Instanzen nach dem neuen Schema ausgeführt werden, während bereits aktive Instanzen nach dem alten Schema weiterlaufen. Dies ist für Prozesse mit kurzer Dauer ausreichend, wirft aber für lang laufende Prozesse, etwa beim Entwurf technischer Systeme [8] oder bei Behandlungsprozessen im Krankenhaus [12], erhebliche Probleme auf. Hier muss dann ggf. für längere Zeit eine Koexistenz von Instanzen alter und neuer Form und damit ein Durcheinander bei der Prozessabwicklung in Kauf genommen werden. Zudem ist eine Fortführung der Prozesse auf Grundlage des alten Schemas nicht immer akzeptabel, etwa wenn dadurch gesetzliche Vorschriften oder Geschäftsregeln des Unternehmens (z.B. Behandlungsrichtlinien eines Krankenhauses) verletzt werden.

Von Anwenderseite besteht deshalb häufig der Wunsch, die auf Schemaebene definierten Änderungen auch auf die bereits (vielleicht in großer Zahl) laufenden WF-Instanzen zu übertragen. Wir sprechen von der *Propagation* einer WF-Schemaänderung auf laufende WF-Instanzen bzw. von der *Migration* dieser („alten") WF-Instanzen auf das geänderte WF-Schema. Dies bei Bedarf zu können, und zwar ohne dass es dadurch zu Inkonsistenzen oder Fehlern kommt, ist unerlässlich, wenn ein WfMS breit und flexibel einsetzbar sein soll. Kommerzielle WfMS bieten hierfür keine ausreichende Unterstützung. Sie erlauben es entweder gar nicht, die Änderungen eines Schemas auf laufende Instanzen zu propagieren, oder aber dies kann zu Inkonsistenzen und Systemabstürzen führen [26].

Für die Abänderung von WF-Schemata und die Migration von WF-Instanzen auf das neue WF-Schema sind folgende Eigenschaften zu fordern:

- *Korrektheit/Konsistenz:* Eine WF-Instanz darf nur dann auf das geänderte WF-Schema migriert werden, wenn es dadurch in der Folge nicht zu unerwünschten Effekten kommt.
- *Anpassungen im laufenden Betrieb und Effizienz*: Die Propagation von Änderungen eines WF-Schemas auf („alte") WF-Instanzen muss im laufenden Betrieb möglich sein. Dies bedingt, dass die bei Migrationen notwendigen Korrektheitsüberprüfungen und Zustandsanpassungen, selbst bei großer Instanzzahl, effizient durchführbar sind. Beispielsweise sollten hierfür benötigte Sperren auf Instanzen bestimmte Zeitschranken nicht überschreiten.
- *Vollständigkeit:* Es müssen alle Aspekte eines WF-Schemas, die von Änderungen betroffen sein können, berücksichtigt werden. Insbesondere dürfen wichtige Elemente (z.B. Schleifen und Datenflüsse) nicht unberücksichtigt bleiben, nur weil dies die Komplexität der Betrachtungen reduziert.
- *Benutzerfreundlichkeit*: Bei Migrationen sind teure Benutzerinteraktionen (möglichst) zu vermeiden. Dies würde zum einen zu Verzögerungen bei der Ausführung der WF-Instanzen führen, zum anderen wären betroffene Benutzer (z.B. Modellierer bzw. Prozessverantwortliche) vielfach überfordert.

Weitere Anforderungen betreffen den Umgang mit nicht migrierbaren WF-Instanzen, die Einbeziehung von Semantik-Aspekten bei Verträglichkeitsprüfungen, das Zusammenspiel von Änderungen auf Typ- und Instanz-Ebene (im Kontext von Ad-hoc-Änderungen) sowie Änderungen anderer Bestandteile prozessorientierter Informationssysteme (z.B. des Organisationsmodells). Auf diese Aspekte wird in diesem Beitrag aus Platzgründen nicht eingegangen (siehe [33,39]).

Die Evolution von WF-Schemata weist gewisse Parallelen zu Schemaänderungen in DBMS auf [4,10,11]. Die Probleme sind relativ ähnlich, wenn man nur die Abbildung der Elemente (Knoten, Kanten) des alten WF-Schemas auf das neue WF-Schema betrachtet. Bezieht man jedoch die Instanzebene (also die laufenden Prozesse) mit ein, ist zu beachten, dass sich jede Instanz in einem anderen Zustand befinden kann. Abhängig vom konkreten Zustand sowie von den angebotenen Migrationsoperationen, kann eine Migration zulässig sein oder nicht. Sowohl für diese Einteilung von Instanzen als auch für die (konkrete) Durchführung von Migrationen sind effiziente Lösungen gefordert. Eine längere Blockierung der (vielleicht sehr vielen) betroffenen WF-Instanzen während der Analyse- und Migrationsphase sollte deshalb vermieden oder zumindest auf die tatsächlich betroffenen WF-Instanzen dieses Typs eingeschränkt werden.

1.2 Beitrag

Im ADEPT-Projekt arbeiten wir seit einigen Jahren an der Entwicklung einer Technologie, die es ermöglichen soll, unternehmensweite Geschäftsprozesse zu modellieren, zu steuern, zu überwachen und sie auf einfache Weise – bei Bedarf auch im laufenden Betrieb – flexibel zu verändern [6, 12,31,29]. In unseren bisherigen Veröffentlichungen haben wir uns mit Ad-hoc-Änderungen einzelner WF-Instanzen und mit der Modellierung planbarer Abweichungen befasst [7, 12,31,32]. Schwerpunkte bildeten dabei die Definition geeigneter Änderungsoperationen (bzw. Graphtransformationsregeln) sowie Korrektheits-, Skalierbarkeits- und Implementationsaspekte. Die wichtigsten Konzepte haben wir prototypisch im ADEPT-WfMS [19] umgesetzt, das mittlerweile auch von Partnern für die Realisierung anspruchsvoller WF-Anwendungen eingesetzt wird [5,27].

Ziel des vorliegenden Beitrags ist die Entwicklung eines theoretisch fundierten und umfassenden Ansatzes für WF-Typänderungen und deren Propagation auf laufende WF-Instanzen. Dazu sind zunächst geeignete, überprüfbare Kriterien erforderlich, durch deren Anwendung sich die Konsistenz von WF-Instanzen auch bei ihrer Migration auf ein geändertes WF-Schema sicherstellen lässt. Dabei müssen die betroffenen WF-Instanzen (bzw. Ausschnitte davon) für eine bestimmte Zeitspanne angehalten werden, um zu verhindern, dass sie in ihrer Ausführung weiterschreiten und deshalb der Zustand, der als Grundlage für die Analysen herangezogen wird, nicht mehr aktuell ist. Für die meisten Anwendungen ist es jedoch nicht akzeptabel, dass die zu Analyse- und Anpassungszwecken nötigen Sperren auf laufenden Instanzen zu lange gehalten werden. Deshalb stellen wir in diesem Beitrag erstmals einfache Korrektheitsprüfungen für WF-Instanzmigrationen vor, die sich durch eine Minimierung der hierfür benötigten Information und Reduzierung der Komplexität erforderlicher Analysen auszeichnen.

Die effiziente Prüfung auf Verträglichkeit ist jedoch nur eine Seite der Medaille. Ebenso wichtig ist die Durchführung der Migrationen selbst. Wir diskutieren hierzu prinzipielle Lösungsansätze und stellen ein Verfahren vor, mit dem sich die bei Instanzmigrationen notwendigen Zustandsanpassungen automatisch und effizient vornehmen lassen. Dabei soll einerseits der Umfang erforderlicher Zustandsneubewertungen auf ein Minimum reduziert werden, andererseits soll im Anschluss wieder ein korrekter Ausführungszustand resultieren. Ein weiteres wichtiges Anliegen dieses Beitrags ist, die vorgestellten Analyseverfahren nicht auf einfache Basiskonstrukte einzuschränken, sondern auch Instanzmigrationen in Verbindung mit Schleifenänderungen und Datenflussanpassungen zu betrachten. Für all diese Fälle werden wir in diesem Beitrag einfach überprüfbare, präzise Bedingungen angeben.

Kapitel 2 behandelt wichtige Grundlagen, die für das weitere Verständnis erforderlich sind. In Kapitel 3 definieren wir Kriterien für die Verträglichkeit von Instanzen mit einem geänderten Schema. Wir zeigen, unter welchen Voraussetzungen diese erfüllt sind und wie sie sich effizient überprüfen lassen. Kapitel 4 behandelt die Migration verträglicher WF-Instanzen auf ein geändertes Schema und diskutiert Strategien für den Umgang mit nicht migrierbaren Fällen. In Kapitel 5 vergleichen wir unseren Ansatz zunächst mit generellen Alternativen, bevor wir dann ausgewählte Ansätze aus der Literatur diskutieren. Der Beitrag schließt mit einer Zusammenfassung und einem Ausblick.

2 Grundlagen

Den folgenden Betrachtungen legen wir das von uns entwickelte ADEPT Workflow-Metamodell [29,31] zugrunde. Es ist einerseits ausdrucksmächtig genug, um reale Prozesse abbilden zu können [12], andererseits sind die beschreibbaren WF-Schemata sowohl für Entwerfer als auch Anwender gut verständlich. Modelliert werden können alle relevanten Aspekte eines Arbeitsprozesses, wie Kontroll- und Datenflüsse, Bearbeiterzuordnungen, zeitliche Abhängigkeiten oder planbare Abweichungen [6,30,31]. Darüber hinaus hat sich das ADEPT-Metamodell im Zusammenhang mit anderen wichtigen Aspekten, wie statischen und dynamischen Modellanalysen [29], Ad-hoc-Änderungen einzelner WF-Instanzen [31] oder verteilter Ausführung von Workflows [6] sehr gut bewährt. Auch das Zusammenspiel dieser Aspekte haben wir eingehend untersucht [7,19]. Wir werden im Folgenden darlegen, dass das ADEPT-Metamodell auch für die Evolution von WF-Schemata und die kontrollierte Migration von Instanzen gut geeignet ist.

2.1 WF-Modellierung und -Ausführung

Die Spezifikation eines Arbeitsprozesses erfolgt durch graphische Modellierung des WF-Schemas. Dieses legt u.a. die Ausführungsreihenfolgen und -bedingungen von Aktivitäten fest. Intern werden solche Kontrollflüsse durch attribuierte WF-Graphen repräsentiert, die sich durch unterscheidbare Knoten- und Kantentypen auszeichnen. Dieser Ansatz erleichtert zum einen die (effiziente) Analyse von WF-Schemata, zum anderen ist er für die interpretative Ausführung der WF-Graphen nützlich [29]. Zur formalen Repräsentation eines Schemas `S` und zur präzisen Definition von Schemaänderungen verwenden wir eine mengenbasierte Darstellung `S = (N, E, ...)`, wobei `N` die Knotenmenge und `E` die Kantenmenge von `S` beschreibt.

Jeder Kontrollkante `e` wird ein Kantentyp `ET(e)` aus `EdgeTypes = {CONTROL_E, SYNC_E, LOOP_E}` zugeordnet. Dabei definiert `CONTROL_E` „normale" Reihenfolgebeziehungen, `SYNC_E` „wartet-auf"-Beziehungen zwischen Aktivitäten paralleler Ablaufzweige und `LOOP_E` Schleifenrücksprünge [31]. Des Weiteren besitzt jeder Knoten `n` einen Typ `NT(n)` aus der Menge `NodeTypes = {STARTFLOW, ENDFLOW, ACTIVITY, STARTLOOP, END-LOOP, AND-Split, AND-Join, XOR-Split, XOR-Join}`. Auf Grundlage dieser Modellelemente können Sequenzen, Parallel-Verzweigungen (`AND-Split, AND-Join`), Alternativ-Verzweigungen (`XOR-Split, XOR-Join`) und Schleifen (`STARTLOOP, ENDLOOP`) modelliert werden. In unserem Ansatz erfolgt dies über ein Blockkonzept, das wir auch in diesem Beitrag verwenden. Die angestellten Betrachtungen lassen sich unter gewissen Voraussetzungen aber auch auf andere WF-Ausführungsmodelle übertragen.

Ausgehend von einem WF-Schema `S` können neue WF-Instanzen erzeugt und gestartet werden. Damit der WF-Server

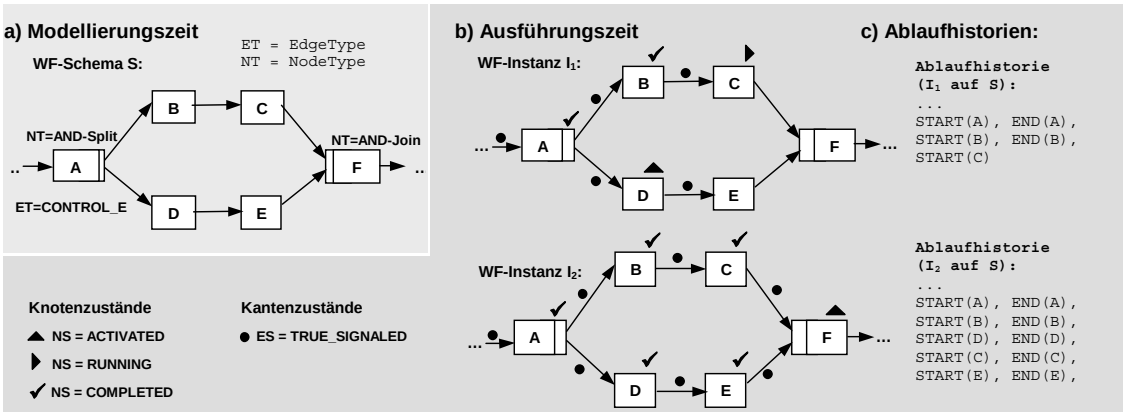


Abb. 1. Modellierung und Ausführung von Workflows in ADEPT

die für eine bestimmte Instanz zur Ausführung anstehenden Aktivitäten ermitteln kann, benötigt er entsprechende Zustandsinformationen. Dazu werden auf Instanzebene jedem Knoten `n` und jeder Kante `e` Zustandsmarkierungen `NS(n)` bzw. `ES(e)` zugeordnet. Ihre Festlegung erfolgt nach wohl definierten Regeln (siehe später), wobei die Markierungen durchlaufener Bereiche (mit Ausnahme von Schleifenrücksprüngen) erhalten bleiben. Zusätzlich werden Knoten und Kanten der Ablaufpfade, die nicht mehr durchlaufen werden, als abgewählt markiert.[1] Der Zustand einer einzelnen Aktivität ist initial `NOT_ACTIVATED` und geht bei ihrer Aktivierung in `ACTIVATED` über. Bei manuellen Aktivitäten werden dann entsprechende Arbeitsaufträge in die Arbeitslisten potenzieller Bearbeiter eingetragen. Wird eine aktivierte Aktivität zur Bearbeitung ausgewählt und gestartet, geht sie in den Zustand `RUNNING` über. Dabei werden entsprechende Arbeitsaufträge aus den Arbeitslisten anderer Benutzer entfernt. Bei erfolgreicher Beendigung erhält die Aktivität den Status `COMPLETED`. Steht dagegen fest, dass sie in der Folge nicht mehr ausführbar ist, wird ihr der Zustand `SKIPPED` zugewiesen. Kanten wird initial der Zustand `NOT_SIGNALED` zugeordnet. Im Verlauf der Ausführung werden sie entweder mit `TRUE_SIGNALED` oder `FALSE_SIGNALED` markiert.

Grundlegend ist auch die *Ablaufhistorie* einer WF-Instanz (vgl. Abb. 2), in welcher unter anderem Informationen zum Start und Ende von Aktivitäten protokolliert werden. Präziser ausgedrückt, schreibt jede Aktivität beim Übergang in den Zustand `RUNNING` einen Starteintrag, beim Übergang in den Zustand `COMPLETED` einen Endeintrag in die Ablaufhistorie.

2.2 Definition und Änderung von WF-Schemata

Eine zentrale Forderung bei der Erstellung und Änderung von WF-Schemata ist, dass die Korrektheit und Konsistenz von Schema- und Instanzdaten zu jedem Zeitpunkt gewährleistet sind. Zu diesem Zweck führt das ADEPT-WfMS schon zur Modellierzeit eine Vielzahl an Korrektheitsprüfungen durch. Insbesondere müssen sowohl für laufende als auch für zukünftige WF-Instanzen unerwünschte Effekte wie Daten-Inkonsistenzen, Verklemmungen oder fehlerhafte Aufrufe von Aktivitätenprogrammen ausgeschlossen werden [29]. Notwendige Voraussetzung hierfür ist, dass nach Modifikation eines (korrekten) Quell-Schemas `S` wieder ein korrektes Ziel-Schema `S'` resultiert. Für diesen statischen Fall wird zunächst nur die korrekte Transformation des WF-Schemas (ohne Instanzen) betrachtet. Sie bildet nicht den Fokus dieser Arbeit, ist aber eine wichtige Voraussetzung für die korrekte Migration von Instanzen.

In unserem Ansatz kommen wir dem nach, indem der Modellierer auf semantisch hoher Ebene ein WF-Schema `S` mittels eines syntaxgesteuerten WF-Editors sowie einer vollständigen Menge von Modifikationsoperationen (z. B. für das Einfügen einer Aktivität zwischen zwei Knotenmengen oder das Löschen von Schleifenblöcken) korrekt abändern kann [31, 29]. Die Korrektheit für den statischen Fall wird durch formale Vor- und Nachbedingungen in Bezug auf die Anwendung dieser Änderungsoperationen sichergestellt. Intern lassen sich solche Operationen auf einfache Einfüge- und Löschoperationen von Knoten und Kanten abbilden, so dass wir uns in diesem Beitrag weitgehend auf diese Elementaroperationen beschränken werden.

3 Effiziente Verträglichkeitsprüfungen bei Änderungen des WF-Schemas

Fokus dieses Abschnitts bildet die grundlegende Frage, anhand welcher Informationen das WfMS überprüfen soll, ob Änderungen eines WF-Schemas korrekt auf laufende WF-Instanzen propagiert werden können. Dazu definieren wir in Abschnitt 3.1 ein allgemeingültiges, formales Kriterium zur Feststellung der Verträglichkeit von WF-Instanzen mit einem geänderten WF-Schema. Wie sich dieses Kriterium in der Praxis effizient überprüfen lässt, zeigen die Abschnitte 3.2 (für Kontrollfluss-Änderungen) und 3.3 (für Datenfluss-Änderungen).

Wie das nachfolgende Beispiel zeigt, darf eine Änderungspropagation niemals unkontrolliert erfolgen. Ansonsten können im Verlauf der weiteren WF-Ausführung ernsthafte Probleme wie Verklemmungen oder fehlerhafte Aktivitätenprogrammaufrufe resultieren.

[1] Aufgrund der Struktureigenschaften von ADEPT WF-Graphen können die aktuellen Knotenmarkierungen einer WF-Instanz auch aus den Zuständen der gerade bearbeiteten Aktivitäten sowie den Verzweigungsentscheidungen bereits beendeter XOR-Split- und Schleifenendknoten abgeleitet werden.

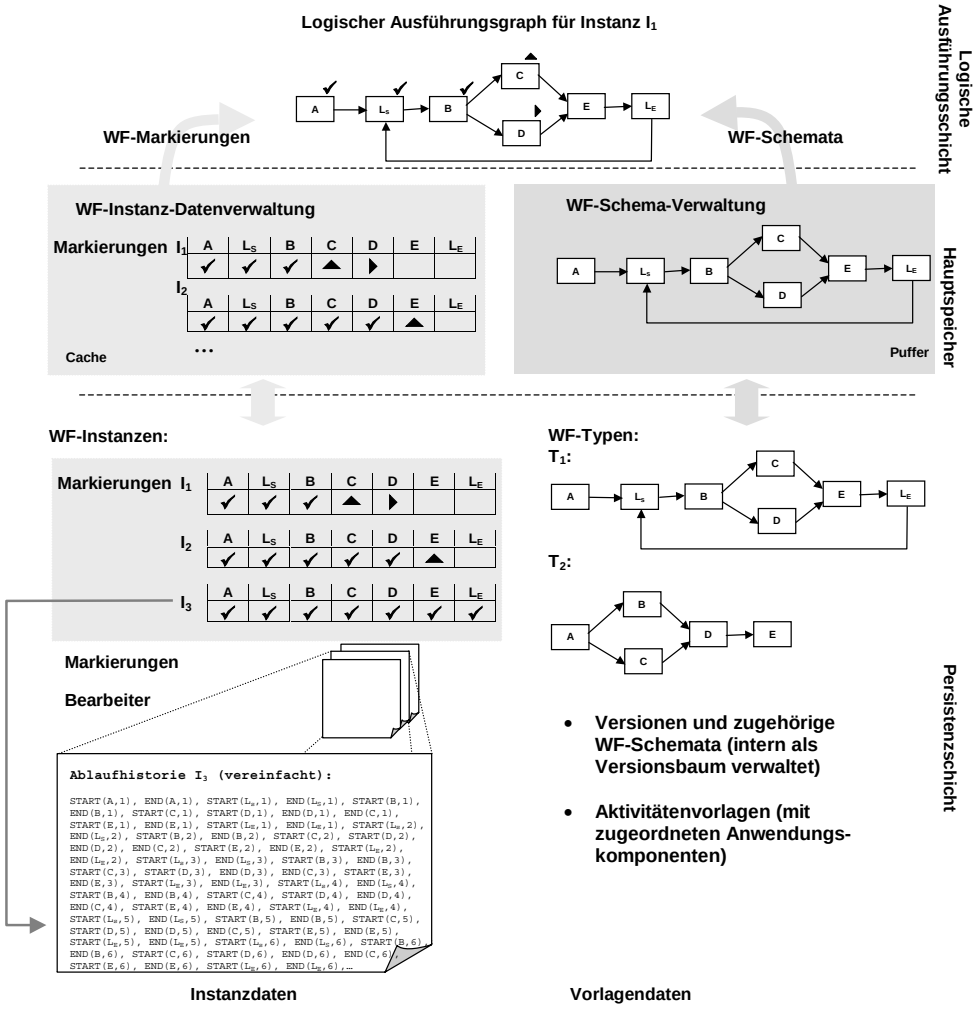


Abb. 2. Schema- und Instanzverwaltung in WfMS (vereinfacht)

Beispiel In das Schema S aus Abb. 3a werden zwei neue Schritte und eine Datenabhängigkeit zwischen ihnen eingefügt, so dass wieder ein korrektes Schema S′ resultiert. Abb. 3b zeigt von S abgeleitete Instanzen I_1 und I_2. Bei Propagierung der Schemaänderung auf I_1 würde der Schritt Allergietest durchführen in einen bereits durchlaufenen Bereich eingefügt, wodurch das Ausgabedatum Allergietyp vor Aktivierung des datenabhängigen Schritts Arznei verordnen nicht mehr geschrieben werden könnte. Liest die neu eingefügte Aktivität Arznei verordnen dieses Datum obligat, bleiben Eingabeparameter des zugeordneten Aktivitätenprogramms unversorgt, was zu schwerwiegenden Konsequenzen führen kann (z. B. Programmabstürze). Für I_2 dagegen könnte Allergietest durchführen das Ausgabedatum schreiben, so dass die Aktivität Arznei verordnen korrekt versorgt werden kann.

3.1 Verträglichkeit von WF-Instanzen mit dem geänderten WF-Schema

Angenommen auf Typebene wird nun ein korrektes WF-Schema S in ein wiederum korrektes Schema S′ transformiert. Dann dürfen die vorgenommenen Änderungen nur dann auf eine Instanz I von S propagiert werden, wenn I auch auf Grundlage von S′ korrekt ausführbar ist, d.h. wenn die WF-Instanz I - entsprechend ihrem bisherigen Verlauf - auch gemäß S′ ausgeführt hätte werden können und dabei dieselben Effekte auf WF-Variablen (bzw. WF-Daten) bewirkt hätte (insbesondere darf es infolge einer Migration nicht zu einer „Verfälschung" der Vergangenheit kommen). Wir nennen I dann *verträglich* mit S′.

Die spannende Frage ist nun, unter welchen Bedingungen für WF-Instanzen zuverlässige Verträglichkeitsaussagen getroffen werden können und welche Informationen für entsprechende Überprüfungen (minimal) benötigt werden. Intuitiv ist klar, dass man hierfür gewisse Daten zum bisherigen Verlauf der WF-Ausführung benötigt. Im ADEPT-WfMS werden, wie in einigen anderen Ansätzen auch, Informationen zum Start und Ende von Aktivitätenbearbeitungen in einer Ablaufhistorie verwaltet, die u.a. auch für das korrekte Rücksetzen von WF-Instanzen im Fehlerfall genutzt wird. Bei Verwendung der Ablaufhistorie $\mathcal{A}$ einer WF-Instanz I kann Verträglichkeit mit dem geänderten WF-Schema S′ zugesichert werden, wenn $\mathcal{A}$ auch bei Ausführung auf S′ erzeugbar gewesen wäre. Formal:

Definition 3.1 (Verträglichkeits-Kriterium). *Sei* $S = (N, E, ...)$ *ein korrektes WF-Schema und I eine davon abgeleitete WF-Instanz mit Ablaufhistorie* $\mathcal{A} =< e_0, \ldots, e_t >$ *und Markierung* M_t*;* $e_0, \ldots, e_t$ *entsprechen dabei den Start-/End-ereignissen zu laufenden bzw. beendeten Aktivitäten, deren se-*

quenzielle Anwendung auf I die WF-Instanz von ihrer Anfangsmarkierung M_0, über Zwischenmarkierungen $M_1, \ldots, M_{t-1}$, in ihre aktuelle Markierung M_t überführt hat (kurz: $M_0[e_0 > M_1[e_1 > \ldots M_{t-1}[e_t > M_t$).[2]

S werde nun durch eine Änderung Δ auf das (korrekte) WF-Schema S' transformiert. Sei M'_t die Markierung des Ausführungsgraphen von I, der sich ergibt, wenn Δ auf I propagiert wird und anschließend eine Neubewertung des Zustands des Ausführungsgraphen (nach wohl definierten Regeln) durchgeführt wird. Dann ist I genau dann verträglich mit S', wenn dieselbe Folge von Ereignissen $e_0, \ldots, e_t$, ausgehend von einer Anfangsmarkierung M'_0, auf WF-Schema S' anwendbar ist und im Anschluss wieder die konsistente Markierung M'_t resultiert. Formal:

I ist verträglich mit S' $:\Leftrightarrow M'_0[e_0 >, \ldots, [e_t > M'_t$

Beispiel (Verträglichkeits-Kriterium). Für Instanz `I1` in Abb. 3b ist der Schritt `Patient aufklären` beendet und der Schritt `Patient untersuchen` gestartet. Dementsprechend enthält die Ablaufhistorie Informationen zum Start bzw. Ende dieser Aktivitäten. Wird nun auf Schemaebene die Aktivität `Allergietest durchführen` vor `Patient vorbereiten` eingefügt, könnte für `I1` die bisherige Ablaufhistorie nicht mehr auf dem geänderten Schema `S'` erzeugt werden. Damit ist `I1` nicht verträglich mit `S'` und kann demzufolge nicht auf `S'` migriert werden. Im Falle von Instanz `I2` sind noch keine Einträge in die Ablaufhistorie geschrieben worden, so dass `I2` verträglich mit `S'` ist.

Obiges Kriterium ist unabhängig vom verwendeten WF-Ausführungsmodell und bildet deshalb eine gute Ausgangsbasis für unsere weiteren Betrachtungen. Insbesondere kann eine WF-Instanz, die gemäß dieser Definition verträglich mit einem geänderten WF-Schema ist, problemlos migriert werden. Darüber hinaus bleiben orthogonale Systemfunktionen (z.B. semantisches Rollback) unverändert bestehen, eine Eigenschaft, die für die Implementation adaptiver WfMS essenziell ist. Jedoch stellt sich die Frage, wie sich für (eine große Zahl von) WF-Instanzen Verträglichkeit mit dem neuen Schema effizient feststellen lässt. Obwohl es in der Literatur bereits einige Vorschläge für die Evolution von WF-Schemata gibt (z.B. [9,22,24,34,37]) findet man hierzu – von einigen Ausnahmen abgesehen (z.B. [24]) – keine detaillierten Aussagen. Ebenso wenig wurde bisher versucht, den Umfang der notwendigen Überprüfungen und der hierfür benötigten Daten geeignet einzuschränken, etwa durch Einbeziehung der speziellen Semantik von Änderungsoperationen (z.B. Verträglichkeit bei Einfüge- und Löschoperationen) oder durch geeignete Verdichtung von Informationen aus der Ablaufhistorie. Schließlich ist bei vielen Ansätzen unklar, wie für verträgliche Instanzen die Migration auf das neue Schema konkret erfolgen soll. In der Praxis führt dies dazu, dass die vorgeschlagenen Lösungskonzepte zu restriktiv sind, sich zu komplex für Benutzer darstellen oder aber zu „lückenhaften" Implementierungen führen. So werden von einigen Ansätzen viele WF-Instanzen von einer Migration ausgeschlossen, für die eine Änderungspropagation problemlos möglich wäre (etwa bei Änderungen in Verbindung mit Schleifen). Einige Ansätze reduzieren die Komplexität dadurch, dass sie auf bestimmte, für die Praxis wichtige Modellelemente (z.B. Schleifen) gänzlich verzichten. Entsprechend eingeschränkt ist ihr Einsatzspektrum.

Definition 3.1 bildet die formale Grundlage für die nachfolgenden Betrachtungen. Allerdings wäre es naiv, für jede Instanz zu untersuchen, ob ihre komplette Ablaufhistorie auf dem geänderten WF-Schema „nachgespielt" werden kann. Der Umfang dieser Historie kann bereits für einzelne Instanzen sehr groß sein. So müssen alle relevanten Ereignisse (z.B. zum Start und zur Beendigung von Aktivitäten) protokolliert werden (vgl. Abb. 6). Daneben umfasst ein Historieneintrag auch Informationen zu Bearbeitern oder Zeitpunkten. Sie werden benötigt, um im Fehlerfall ein korrektes Rücksetzen der WF-Instanz in einen früheren Bearbeitungszustand durchführen zu können. Die Historiengröße kann in Verbindung mit Schleifen beträchtlich sein, da für jede Iteration die Informationen zur Ausführung der Schleifen-Aktivitäten protokolliert werden. Darüber hinaus wird die Ablaufhistorie in den meisten WfMS nicht im Hauptspeicher gehalten, sondern auf Externspeicher ausgelagert (siehe Abb. 2). Dies ist in der Regel ausreichend, da Zugriffe auf die komplette Ablaufhistorie nur in Verbindung mit eher seltenen Operationen (z.B. Rollback, Crash-Recovery) erforderlich sind.

Nachfolgend zeigen wir, dass für Verträglichkeitsprüfungen in der Regel nicht die kompletten Historiendaten notwendig sind, sondern in der Mehrzahl der Fälle wesentlich weniger Informationen ausreichen. Des Weiteren werden wir im Zusammenhang mit Schleifen sehen, dass hier Informationen zu aktuellen Knotenmarkierungen für die Verträglichkeitsprüfungen genügen, d.h. auf Historiendaten früherer Schleifeniterationen muss bei Verträglichkeitsprüfungen nicht zurückgegriffen werden. Dies ist für die Implementierung adaptiver WfMS sehr hilfreich.

In den Abschnitten 3.2 und 3.3 stellen wir einfach überprüfbare Verträglichkeitsbedingungen vor, die auf Markierungsinformationen basieren, so dass bei der Änderungspropagation aufwändige Zugriffe auf die komplette Ablaufhistorie entfallen. Außerdem werden zum ersten Mal auch Änderungen auf Schleifen und Datenflüssen systematisch betrachtet.

3.2 Verträglichkeit bei Kontrollflussänderungen

Gegeben sei ein korrektes WF-Schema `S` mit davon abgeleiteten, laufenden WF-Instanzen `I1`, ..., `In`. `S` werde durch eine Änderung Δ auf ein korrektes WF-Schema `S'` transformiert. Wir wollen zeigen, dass die Verträglichkeit einer WF-Instanz `Ik` mit `S'` zugesichert werden kann, wenn `Ik` vor der Migration bestimmte Zustandsmarkierungen aufweist. Dadurch garantieren wir die Konsistenz der betroffenen WF-Instanz auch nach Propagation der Schemaänderungen. Wir werden zeigen, dass solche Vorbedingungen – ganz abgesehen vom Umfang der benötigten Informationen – einfach und effizient überprüfbar sind. Aus Darstellungsgründen unterscheiden wir im Folgenden zwischen azyklischen WF-Graphen (*Graphklasse 1*) und WF-Graphen mit Schleifen (*Graphklasse 2*), wobei für Letztere das Verträglichkeitskriterium aus Def. 3.1 noch weniger restriktiv gestaltet werden muss. Dabei beschränken wir uns zunächst auf Kontrollflussaspekte und klammern Datenflüsse zwischen Aktivitäten (siehe Abschnitt 3.3) noch aus.

[2] $M_i[e_i > M_{i+1}$ bedeutet, dass M_i durch Anwendung des Ereignisses e_i in M_{i+1} überführt wird.

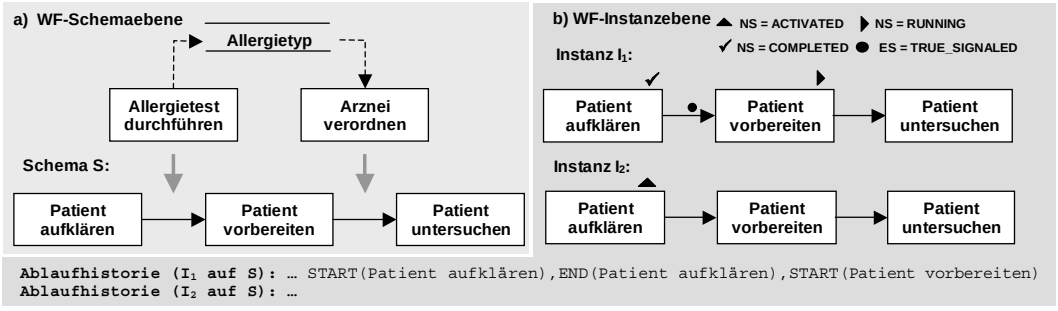


Abb. 3. Einfügen von Aktivitäten

Zusätzlich differenzieren wir zwischen additiven, subtraktiven und ordnungsverändernden Operationen, um präzise angeben zu können, welche Informationen für Verträglichkeitsprüfungen jeweils benötigt werden.

3.2.1 WF-Graphen ohne Schleifen (Graphklasse 1)

Für WF-Graphen der Klasse 1 lassen sich sequenzielle, parallele und alternative Ausführungspfade modellieren. Beispiele zeigen Abb. 1 und 4. Bei Verwendung von XOR-Verzweigungen wird es Pfade geben, für die sich bei der Ausführung (z.B. abhängig von WF-relevanten Daten) ergibt, dass sie nicht bearbeitet werden sollen. Sie werden entsprechend als SKIPPED markiert. Beispielsweise wurde bei Ausführung von Instanz I_1 in Abb. 4b der Pfad mit Selektions-Code sc1 gewählt[3] (vgl. Historie aus Abb. 4c), weshalb die Aktivitäten der anderen Ablaufzweige (D, E, F und G) als SKIPPED markiert sind. Wichtig ist, dass solche Aktivitäten keine Einträge in die Ablaufhistorie schreiben, was durch Abb. 4c illustriert wird.

Tabelle 1 zeigt für additive und subtraktive Änderungen, welche formalen Bedingungen erfüllt sein müssen, damit eine WF-Instanz I verträglich mit dem geänderten WF-Schema S′ ist. Dabei wird vorausgesetzt, dass das aus der Änderung hervorgegangene WF-Schema S′ korrekt ist, insbesondere muss S′ wieder ein Vertreter der Graphklasse 1 sein. Wir diskutieren die angegebenen Bedingungen und ihre Vorzüge im Folgenden anhand von Beispielen, verzichten aus Platzgründen aber weitgehend auf formale Darstellungen und Beweise (siehe [33]).

a) Additive Änderungsoperationen. Gegeben sei ein WF-Schema der Graphklasse 1 und eine WF-Instanz I auf S. S werde durch Einfügen einer Aktivität oder Hinzunahme einer Kontroll- bzw. Sync-Kante in das WF-Schema S′ (der Graphklasse 1) transformiert. Bei Einfügen einer Aktivität kommen zusätzlich Kontrollabhängigkeiten hinzu, um diese Aktivität in den WF-Kontext einzubetten. Tabelle 1.1 fasst zusammen, unter welchen Bedingungen I verträglich mit S′ ist. Hieraus geht auch präzise hervor, welche Markierungen oder Historiendaten für die Verträglichkeitsprüfung benötigt werden. Beim Einfügen von Aktivitäten kann Verträglichkeit demnach zugesichert werden, wenn sich die (direkten) Nachfolger des neu eingefügten Schritts n_{insert} aktuell in einem der beiden Zustände ACTIVATED oder NOT_ACTIVATED befinden. Knoten im Zustand NOT_ACTIVATED sind generell unproblematisch. Bereits aktivierte Knoten werden zwar schon in Arbeitslisten zur Auswahl angeboten, sind aber noch nicht gestartet worden. Insbesondere haben sie noch keine Einträge in die Historie geschrieben, so dass sie bei Bedarf problemlos aus den Arbeitslisten entfernt werden können.

Bei alternativen Ablaufzweigen kann Verträglichkeit auch für den Fall zugesichert werden, dass ein direkter Vorgänger oder Nachfolger (oder beide) von n_{insert} im Quell-Schema S aktuell die Markierung SKIPPED besitzt oder n_{insert} in einen leeren Teilzweig, dessen Kante als FALSE_SIGNALED markiert wurde, eingefügt wird. Für Aktivitäten, die als SKIPPED markiert sind, werden in der Historie keine Einträge protokolliert. Damit hat das Einfügen von Knoten in abgewählte Teilzweige keinen Effekt auf die Verträglichkeit einer WF-Instanz mit dem neuen WF-Schema. Beispielsweise kann für I_2 aus Abb. 4b eine neue Aktivität X ohne Probleme vor C eingefügt werden. Der Status von X wird dann ebenfalls als SKIPPED bewertet, so dass sich die Historie aus Abb. 4c auch auf dem geänderten Schema erzeugen lässt.

Für das Einfügen einer Kontroll-/Sync-Kante kann man Verträglichkeit zusichern, wenn der Zielknoten der Kante noch nicht gestartet wurde (vgl. Tab. 1.1). In Abb. 1b etwa kann D $\rightarrow$ C nicht in I_1 eingefügt werden, da sich C bereits im Zustand RUNNING befindet, der Knoten D aber noch nicht beendet wurde. Dies ist auch aus der Historie von I_1 (vgl. Abb. 1c) ersichtlich: C hat seinen Starteintrag geschrieben, ohne dass ein Eintrag zu D existiert. Folglich lässt sich diese Historie auf dem geänderten Schema nicht mehr erzeugen. Jedoch können Kontroll- und Sync-Kanten auch noch dann in eine WF-Instanz eingefügt werden, wenn ihr Quell- und Zielknoten bereits beendet sind, diese aber ihre Historieneinträge in der Reihenfolge der neuen Kontrollabhängigkeit geschrieben haben. Zur Feststellung sind allerdings gewisse Informationen aus der Ablaufhistorie nötig. Eine Einfügung von D $\rightarrow$ C in I_2 etwa ist möglich. Zwar besitzen hier sowohl Quell- als auch Zielknoten den Status COMPLETED, jedoch haben C und D ihre Historieneinträge in der „richtigen" Reihenfolge geschrieben (siehe Abb. 1c).

Das Einfügen einer Sync-Kante $n_{src} \rightarrow n_{dest}$ zwischen bisher parallel ausführbaren Knoten n_{src} und n_{dest} ist unkritisch (vgl. Tab. 1.1), solange der Zielknoten n_{dest} einen der Zustände NOT_ACTIVATED, ACTIVATED oder SKIPPED besitzt. Grund ist, dass n_{dest} in diesem Fall (noch) keinen Historieneintrag geschrieben hat. Andernfalls gilt NS(n_{dest}) $\in$ {RUNNING, COMPLETED} und die Verträglichkeit von I mit dem neuen Schema ist nicht immer gewährleistet. Dies ist

[3] In ADEPT kann ein Selektions-Code von einem Aktivitätenprogramm geliefert werden, er kann aber auch Ergebnis einer Prädikatevaluation sein (entsprechende Prädikate sind dann dem XOR-Splitknoten zugeordnet).

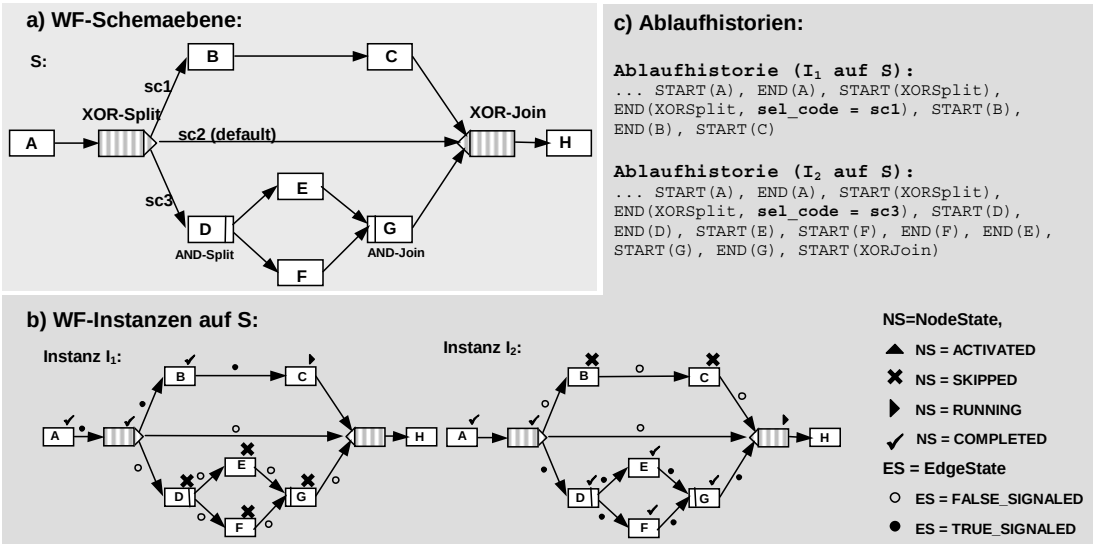


Abb. 4. WF-Graph mit Parallel- und Alternativ-Verzweigung (Graphklasse 1)

Tabelle 1. Änderungsoperationen für Graphklasse 1

a) Additive Änderungsoperationen(Graphklasse 1)

Einfügen von	*Eine Instanz* I *mit Historie* $\mathcal{A}$ *ist verträglich mit* S′ ⇔
Aktivitätenknoten n_{insert}	[$\forall$ n ∈ {x ∈ N | $n_{insert} \rightarrow$ x ∈ E'}: NS(n) ∈ {NOT_ACTIVATED, ACTIVATED, SKIPPED}] ∨ [n_{insert} wird in einen abgewählten Teilzweig einer XOR-Verzweigung eingefügt]
Kontrollkante $n_{src} \rightarrow n_{dest}$	NS(n_{dest}) ∈ {NOT_ACTIVATED, ACTIVATED, SKIPPED} ∨ [NS(n_{src}) = COMPLETED ∧ NS(n_{dest}) ∈ {RUNNING, COMPLETED}) mit ($\exists e_i$ = END(n_{src}), e_j = START(n_{dest}) ∈ $\mathcal{A} \wedge i < j$)]
Sync-Kante $n_{src} \rightarrow n_{dest}$	NS(n_{dest}) ∈ {NOT_ACTIVATED, ACTIVATED, SKIPPED} ∨ (NS(n_{src}) = COMPLETED ∧ NS(n_{dest}) ∈ {RUNNING, COMPLETED}) mit ($\exists e_i$ = END(n_{src}), e_j = START(n_{dest}) ∈ $\mathcal{A} \wedge i < j$)) ∨
(n_{src} und n_{dest} bisher parallel angeordnet)	(NS(n_{src}) = SKIPPED ∧ NS(n_{dest}) ∈ {RUNNING, COMPLETED}) mit $\forall$ n ∈ $N_{critical}$ mit NS(n) ≠ SKIPPED: $\exists e_i$ = START(n_{dest}), e_j = END(n) ∈ $\mathcal{A}$ mit $j < i$), wobei $N_{critical}$ = (c_pred*(S, n_{src}) ¬ c_pred*(S, n_{dest})) (c_pred*(S, n) bezeichnet dabei die Menge aller direkten und indirekten Vorgängerknoten von n bzgl. Kanten des Typs CONTROL_E im WF-Schema S)

b) Subtraktive Änderungsoperationen (Graphklasse 1)

Löschen von	*Eine Instanz* I *ist verträglich mit* S′ ⇔
Aktivitätenknoten $n_{delete} \in N$	NS(n_{delete}) ∈ {NOT_ACTIVATED, ACTIVATED, SKIPPED}
Kontroll- oder Sync-Kante $n_{src} \rightarrow n_{dest}$	keine Bedingung

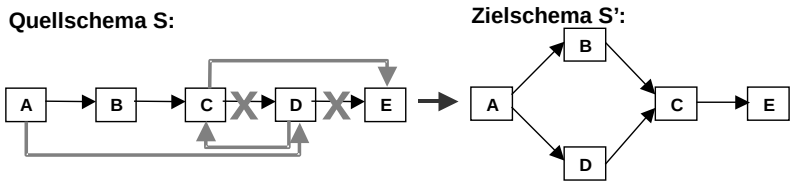


Abb. 5. Verschieben von Aktivitäten

nur der Fall, wenn n_{src} bereits beendet ist und seinen Endeintrag in der Historie vor dem Starteintrag von n_{dest} geschrieben hat. Gilt dagegen NS(n_{src}) = SKIPPED, ist I nur dann verträglich mit S', wenn alle Vorgängerknoten von n_{src} mit Status ungleich SKIPPED ihre Endeinträge in der Ablaufhistorie vor dem Starteintrag von n_{dest} geschrieben haben.

b) Subtraktive Änderungsoperationen. Für subtraktive Änderungen, bei denen Aktivitäten bzw. Kanten aus dem WF-Schema gelöscht werden, sind Verträglichkeitsprüfungen vollständig auf Grundlage von Zustandsmarkierungen durchführbar. Intuitiv ist klar, dass man nur solche Aktivitäten löschen darf, die noch keinen Historieneintrag geschrieben haben (vgl. Tab. 1.2). Bei nicht aktivierten Knoten sind keine Anpassungen notwendig, ansonsten müssen ggf. Arbeitslisteneinträge gelöscht werden, bevor mit der WF-Kontrolle fortgefahren werden kann. Löscht man bspw. Knoten D (mit Status ACTIVATED) aus Instanz I_1 in Abb. 1b, kann die Historie aus Abb. 1c auch auf dem geänderten Schema erzeugt werden. Allerdings müssen bei Migration von I_1 auf das neue Schema die bisherigen Einträge zu D aus Arbeitslisten entfernt werden. Da eine als SKIPPED markierte Aktivität keinen Historieneintrag geschrieben hat, ist hier das Löschen immer unkritisch. So hat in Abb. 4b die als SKIPPED markierte Aktivität E von I_1 keinen Historieneintrag geschrieben (vgl. Abb. 4c). Damit kann diese Historie auch auf dem WF-Schema erzeugt werden, das sich nach Löschen von E ergibt. Beim Löschen von Kanten werden Reihenfolgebeziehungen zwischen Aktivitäten aufgehoben, so dass bzgl. Verträglichkeit keine Status-Bedingungen gestellt werden müssen.

c) Ordnungsverändernde Operationen. Wir betrachten nun ordnungsverändernde Operationen als eine spezielle, für die Praxis sehr wichtige Art von WF-Graphtransformationen. Sie ändern die Anordnungsbeziehungen zwischen Aktivitätenknoten, ohne dass die Knotenmenge selbst modifiziert wird. Entsprechende Änderungen werden etwa notwendig, wenn die Bearbeitungsreihenfolge von Knoten vertauscht werden soll oder - allgemeiner - wenn ein Knoten von seiner aktuellen Position im WF-Graphen an eine andere Stelle verschoben werden soll. Prinzipiell kann man ordnungsverändernde Operationen auf eine Menge elementarer Kanteneinfüge- und Kantenlöschoperationen zurückführen.

Beispiel (Verschieben von Aktivitäten). Schema S in Abb. 5 umfasst sequenziell angeordnete Aktivitäten A, B, C, D und E. Im Folgenden wird eine parallele Anordnung von B und D gewünscht, wie im Zielschema dargestellt. Dazu muss D zunächst aus seinem Kontext im WF-Graphen herausgelöst werden (Entfernen von C → D und D → E). Anschließend wird D durch Einfügen der Kontrollkanten A → D, D → C und C → E parallel zum Knoten B angeordnet.

Kantenlöschoperationen sind, wie bereits gezeigt, unkritisch, da bestehende Reihenfolgebeziehungen zwischen Aktivitäten aufgelöst werden. Beispielsweise besteht nach Entfernen der Kontrollkanten C → D und D → E in Abb. 5 keine Reihenfolgebeziehung mehr zwischen diesen Knoten. Kanteneinfügeoperationen haben keine Auswirkung auf die Verträglichkeit von WF-Instanzen mit dem neuen WF-Schema, wenn sie Reihenfolgebeziehungen zwischen Aktivitäten festlegen, die schon im Quellschema S bestanden oder (transitiv) auch im Zielschema S' weiterhin gelten. Ein Beispiel hierfür ist das Einfügen von C → E in Abb. 5, da C bereits für Instanzen des Schemas S immer vor E ausgeführt wird. Kritisch sind nur solche Einfügeoperationen, die neue Reihenfolgebeziehungen zwischen Aktivitäten definieren, die in S noch nicht enthalten waren, jedoch für das Zielschema S' gelten. In Abb. 5 wird z.B. durch Einfügen von D → C eine neue Reihenfolgebeziehung festgelegt, was die Beachtung des Status von C erfordert. Im Allgemeinen ergeben sich die Voraussetzungen für die Verträglichkeit von I mit S' durch Aggregation der entsprechenden Bedingungen der angewandten Elementaroperationen. Formal:

Instanz I von S mit Historie $\mathcal{A}$ ist verträglich mit S'
⇔
$\forall$ e = $n_{src} \rightarrow n_{dest} \in$ AddedEdges:[4]

NS(n_{dest}) ∈ {NOT_ACTIVATED, ACTIVATED} ∨
(NS(n_{dest}) ∈ {RUNNING, COMPLETED} ∧
NS(n_{src}) = COMPLETED ∧
(($\exists$ e_i = END(n_{src}), e_j = START(n_{dest})) ∈ $\mathcal{A}$ ∧
$i < j$))

Ein Beweis hierzu findet sich in [33]. Ordnungsverändernde Operationen sind ein Beispiel für komplexe Änderungen, die sich aus der Hintereinanderausführung von Elementaroperationen ergeben. Sind die geforderten Verträglichkeitsbedingungen für die Einzeloperationen erfüllt, gilt dies auch für die komplexe Gesamtoperation.

3.2.2 WF-Graphen mit Schleifen (Graphklasse 2)

Im Folgenden erweitere Graphklasse 2 die Graphklasse 1 um die Möglichkeit zur Definition von Schleifen. In ADEPT bildet jede Schleife eines WF-Graphen einen Kontrollblock mit eindeutigen Start- und Endknoten, die über eine Schleifenrücksprungkante verknüpft sind. Schleifenblöcke können geschachtelt sein. Modellintern werden sie durch spezielle Knoten- und Kantentypen (START_LOOP, END_LOOP, LOOP_EDGE) repräsentiert (vgl. Abb. 7). Durch die Unterscheidbarkeit zwischen „normalem" Fortschritt im Kontrollfluss und Schleifenrücksprüngen ist es insbesondere möglich, „gewünschte" Zyklen von „unerwünschten" Zyklen, deren Auftreten zur Laufzeit zu Verklemmungen führt, zu unterscheiden. Außerdem ist zur Laufzeit einfach zu ermitteln,

[4] Menge der in S' neu hinzugekommenen Kanten

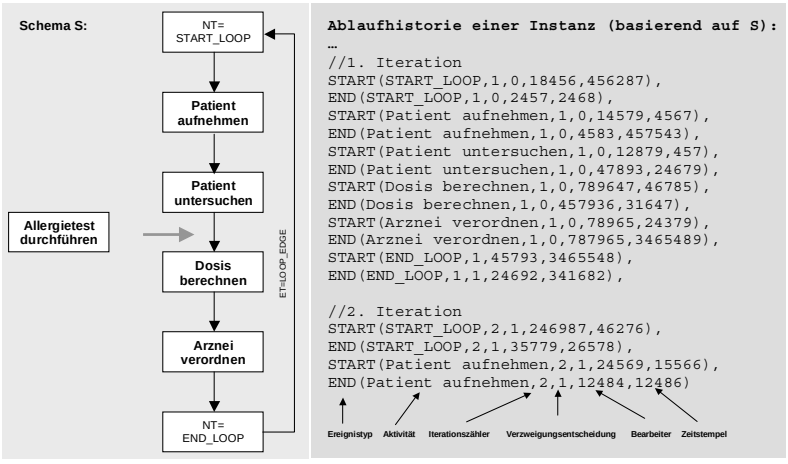


Abb. 6. Ausschnitt aus einem Therapie-Prozess mit Ablaufhistorie

wann es zu einem Schleifenrücksprung kommt. Bei einem solchen Rücksprung werden die Knoten- und Kantenmarkierungen des Schleifenkörpers wieder auf NOT_ACTIVATED bzw. NOT_SIGNALED zurückgesetzt.

Wie erwähnt ist ein zentrales Anliegen unseres Ansatzes, WF-Instanzmigrationen nach Änderungen <u>aller</u> möglichen Modellkonstrukte zu ermöglichen. Das Kriterium aus Def. 3.1 ist hierfür allerdings noch zu restriktiv. Wie das folgende Beispiel zeigt, wäre eine Migration von WF-Instanzen bei schleifenbezogenen Änderungen des WF-Schemas praktisch nie möglich.

Beispiel (Verträglichkeit bei Änderungen innerhalb von Schleifen). Wir betrachten Abb. 6. Angenommen, es ist eine WF-Instanz gegeben, die auf dem Quell-Schema S die dargestellte Ablaufhistorie erzeugt hat. Es ist offensichtlich, dass diese Historie bei Ausführung des durch Einfügen von Allergietest durchführen entstandenen WF-Schemas S′ nicht erzeugbar ist. Für WF-Instanzen von S′ würden nämlich bereits in der ersten Schleifen-Iteration Einträge zum Start und Ende von Allergietest durchführen in die Ablaufhistorie geschrieben. Damit ist eine Migration der WF-Instanz nach dem in Def. 3.1 vorgestellten Kriterium ausgeschlossen, obwohl strukturell und semantisch keine Probleme auftreten würden. Diese Einschränkung ist zum einen unnötig, zum anderen ist sie für viele Anwendungen nicht akzeptabel. Ein Therapieprozess kann z.B. mehrere identische Behandlungszyklen umfassen, zwischen denen größere Zeitabstände liegen. Schemaänderungen müssen hier aus medizinischen und juristischen Gründen auf aktuelle bzw. zukünftige Behandlungszyklen anwendbar sein [35].

Wir wenden uns nun der Frage zu, wann Änderungen eines WF-Schemas mit Schleifen auf bereits laufende WF-Instanzen propagiert werden können. Wie gezeigt, wird bei strikter Anwendung von Def. 3.1 ggf. die Migrationen von Instanzen ausgeschlossen, für die bei der Änderungspropagierung keine inkonsistenten Zustände oder Fehler resultieren würden. Dies schränkt den Benutzer unnötig ein, was den Wunsch nach einer „Auflockerung" des bisherigen Verträglichkeits-Kriteriums aufwirft. Wichtig sind wieder die Sicherstellung der Konsistenz von Modell- und Instanzdaten sowie die effiziente Überprüfbarkeit entsprechender Regeln. Wir benötigen also für Schleifen ein erweitertes Verträglichkeitskriterium. Aus formaler Sicht bieten sich zwei Vorgehensweisen an. Entweder werden alle bisherigen Schleifeniterationen linearisiert, d.h. es erfolgt – logisch gesehen – eine Hintereinanderausführung aller Iterationen des Schleifenkörpers, oder für Schleifenausführungen werden nur diejenigen Historieneinträge berücksichtigt, die in der aktuellen bzw. letzten Schleifeniteration erzeugt wurden. Den folgenden Betrachtungen legen wir die 2. Variante zugrunde.

Definition 3.2 (Iterationsbereinigte Ablaufhistorie). *Sei I eine Instanz eines WF-Schemas S (der Graphklasse 2) mit Ablaufhistorie* $\mathcal{A} =< e_0, \ldots, e_k >$. *Die iterationsbereinigte Ablaufhistorie* $\mathcal{A}_{red}$ *von I entstehe durch Streichen von Einträgen aus* $\mathcal{A}$ *gemäß folgender Regeln:*
Gelte zunächst $\mathcal{A}_{red} := \mathcal{A}$. *Für jede durch ihren Anfangs- und Endknoten* (L_S, L_E) *eindeutig definierte Schleife aus S werden nun alle zugehörigen Historieneinträge gestrichen, die nicht von der aktuellen bzw. letzten Schleifeniteration erzeugt wurden. Formal:*
Falls $\exists e_x \in \mathcal{A}_{red}$ *mit* e_x = START(L_S, i), $i > 0 \wedge$
$\nexists e_y \in \mathcal{A}_{red}$ *mit* e_y = START(L_S, i+1),
dann entferne alle Einträge e = START(n, μ) *bzw.*
e = END(n, μ) *aus* $\mathcal{A}_{red}$, *für die gilt:*
$n \in (c\_succ^*(S, L_S) \cap c\_pred^*(S, L_E))$
$\cup \{L_S, L_E\} \wedge \mu < i$
($c\_succ^*(S, L_S) \cap c\_pred^*(S, L_E)$ *entspricht dabei der Knotenmenge des Schleifenkörpers.)*

Die iterationsbereinigte Ablaufhistorie entsteht durch Entfernen aller Historieneinträge, die von Aktivitäten eines Schleifenkörpers in einer anderen als der letzten (bei beendeten Schleifen) bzw. aktuellen (bei noch laufenden Schleifen) Iteration geschrieben wurden. $\mathcal{A}_{red}$ repräsentiert logisch betrachtet eine WF-Instanz, bei der alle Schleifen maximal einmal durchlaufen wurden. Für WF-Instanzen der Graphklasse 1 gilt trivialerweise $\mathcal{A} = \mathcal{A}_{red}$. Wir geben nun ein modifiziertes Verträglichkeitskriterium für WF-Graphen mit (geschachtelten) Schleifen an, gemäß dem eine Instanz von S genau dann verträglich mit dem aus einer Änderung resultierenden Schema S′ ist, wenn sich die iterationsbereinigte Ablaufhistorie auch auf S′ erzeugen lässt.

Definition 3.3 (Verträglichkeit bei Schleifen). *Sei I eine Instanz eines WF-Schemas S mit Ablaufhistorie* $\mathcal{A} =< e_0, \ldots, e_k >$ *und iterationsbereinigter Ablaufhistorie* $\mathcal{A}_{red} =< e_{i_1}, \ldots, e_{i_n} >$.

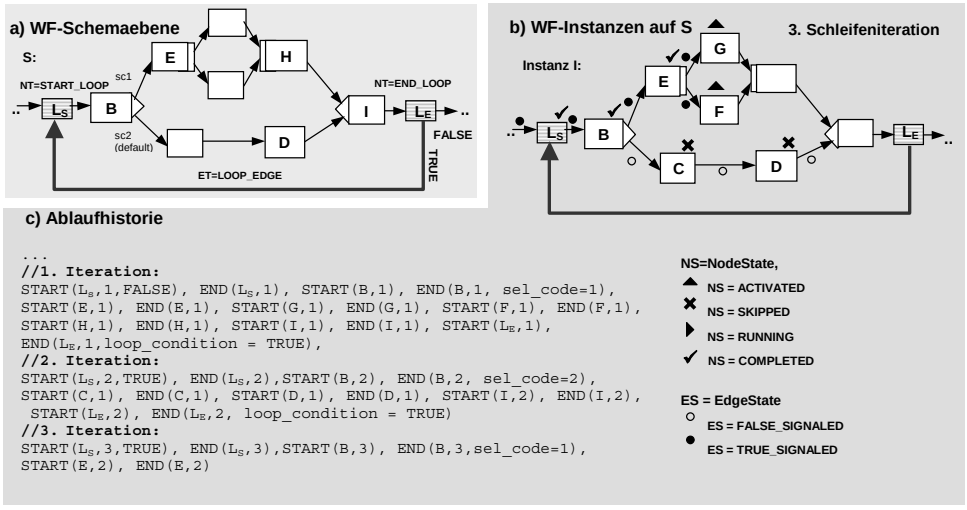


Abb. 7. WF-Graph der Klasse 2

S werde durch eine Änderung Δ auf das (korrekte) WF-Schema S' transformiert. Sei M'_t die Markierung des Ausführungsgraphen von I, der sich ergibt, wenn Δ auf I propagiert wird und anschließend eine Neubewertung des Zustands des Ausführungsgraphen durchgeführt wird. Dann ist I genau dann verträglich mit S', wenn die Folge von Ereignissen $e_{i_1}, \ldots, e_{i_n}$, ausgehend von einer Anfangsmarkierung M'_0, auch auf S' anwendbar ist und im Anschluss wieder eine korrekte Markierung M'_t resultiert. Formal:

I ist verträglich mit S' $:\Leftrightarrow M'_0[e_{i_1} >, \ldots, [e_{i_n} > M'_t$

Beispiel (Verträglichkeit bei Schleifen). Wir betrachten die Instanz aus Abb. 7b. Hier ergibt sich
$\mathcal{A}_{red}$ = <START(L_S,3,TRUE), END(L_S,3), START(B,3), END(B, 3,sel_code=1), START(E,2), END(E,2)>.
Diese Historie ist auch auf dem geänderten WF-Schema, das nach Einfügen von Aktivität X zwischen G und H resultiert, erzeugbar. Damit ist die Instanz verträglich zum geänderten WF-Schema, d.h. die Restriktionen des in Def. 3.1 angegebenen Verträglichkeitskriteriums bestehen nicht mehr.

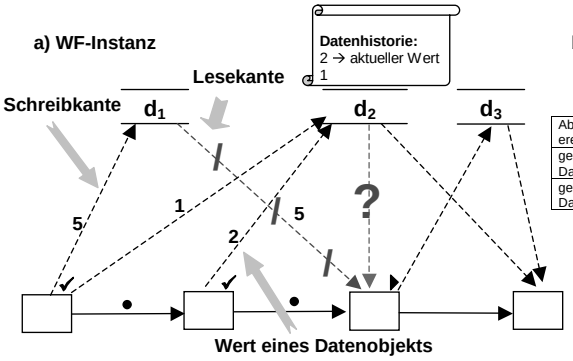
Wir betrachten zunächst die in Abschnitt 3.2.1 diskutierten Schemaänderungen. Um für Instanzen entscheiden zu können, ob sie im Sinne von Def. 3.3 verträglich mit einem geänderten WF-Schema sind, genügt es, dieselben Voraussetzungen zu fordern wie bei WF-Graphen der Klasse 1. Grund dafür ist, dass bei Verwendung der iterationsbereinigten Ablaufhistorie – sie ist für die Verträglichkeitsprüfung maßgebend – Schleifenrücksprungkanten keine Relevanz haben, d.h. man erhält – logisch betrachtet – einen Graphen ohne Schleifen. Der Vollständigkeit halber sei erwähnt, dass ADEPT für die Definition von Schemaänderungen spezielle Operationen anbietet, etwa für das Einfügen, Löschen und Ändern von Verzweigungsblöcken und deren Teilzweigen. Außerdem ist es möglich, Schleifen in ein WF-Schema einzufügen bzw. daraus zu löschen. Darüber hinaus können in ADEPT unter gewissen Voraussetzungen auch neue Schleifen um existierende Blöcke des WF-Schemas eingefügt oder Schleifenrücksprungkanten entfernt werden. Die Vorbedingungen für Verträglichkeitsprüfungen bei Anwendung dieser speziellen Operationen werden in [29] diskutiert.

3.3 Verträglichkeit bei Datenflussänderungen

Unsere bisherigen Betrachtungen haben sich auf Kontrollflussaspekte beschränkt. Im Allgemeinen werden bei WF-Schemaänderungen jedoch auch Anpassungen des modellierten Datenflusses erforderlich. Inwieweit solche Datenflussänderungen die Verträglichkeit von WF-Instanzen mit dem geänderten WF-Schema beeinträchtigen, soll nun behandelt werden.

3.3.1 Datenflussmodellierung und -steuerung in ADEPT

In ADEPT erfolgt der *Datenaustausch* zwischen Aktivitäten über globale Prozessvariablen (sog. *Datenelemente*). Der *Datenfluss* wird modelliert, indem man *Eingabe- und Ausgabeparameter* von Aktivitäten über *Datenkanten* mit diesen *Datenelementen* verknüpft. Dabei wird jeder *Eingabeparameter* mittels genau einer *Lesekante* und jeder *Ausgabeparameter* über genau eine *Schreibkante* an die *Datenelemente* angebunden. Ein Beispiel zeigt Abb. 8a. Hier schreibt B das *Datenelement* d_2, das nachfolgend von C und D gelesen wird. Die Gesamtheit aller einem WF-Schema zugeordneten *Datenelemente* und *Datenkanten* bezeichnen wir als *Datenfluss-Schema (DF-Schema)*. Für sie bestehen in ADEPT gewisse Korrektheitsforderungen, etwa dass beim Start eines Aktivitätenprogramms stets alle obligaten Eingabeparameter versorgt sein müssen. Zur Ausführungszeit einer WF-Instanz verwaltet ADEPT für jedes Datenelement ein oder mehrere Versionen eines Datenobjekts. Bei einem Schreibzugriff auf ein Datenelement wird stets eine neue Version des Datenobjekts angelegt, d.h. Datenobjekte werden niemals physisch überschrieben. Die Verwaltung der verschiedenen Versionen ist wichtig für das kontextabhängige Lesen von Datenelementen und das korrekte Zurücksetzen im Fehlerfall (für Details siehe [29]). Der Datenkontext einer WF-Instanz umfasst also für jedes Datenelement eine *Datenhistorie*, die diesem Datenelement eine geordnete Liste der Versionen des verwalteten Datenobjekts zuordnet.



b) Erweiterte Ablaufhistorie

Ablauf-ereignisse	START (A)	END (A)	START (B)	END (B)	START (C)
geschriebene Datenelemente	-	(d_1,5) (d_2,1)	-	(d_2,2)	-
gelesene Datenelemente	-	-	-	-	(d_1,5)

Abb. 8. Änderung des DF-Schemas

3.3.2 Änderungen des Datenfluss-Schemas und Verträglichkeit von WF-Instanzen

Änderungen des DF-Schemas können sowohl als begleitende Anpassungen zu Kontrollfluss-Änderungen (z.B. Löschen einer Aktivität mit Entfernen der assoziierten Datenkanten) als auch als eigenständige Operationen durchgeführt werden. Letzteres ist z.B. zur nachträglichen Korrektur von Fehlern bei der Datenflussmodellierung notwendig. Als Operationen für die Abänderung von DF-Schemata unterstützt ADEPT das Einfügen und Löschen von Datenelementen bzw. Datenkanten. Die bisher definierten Verträglichkeitskriterien müssen nun dahingehend erweitert werden, dass sie auch bei Datenfluss-Änderungen sinnvoll anwendbar sind.

Beispiel (Inkonsistentes Lesen). Wir betrachten die Instanz aus Abb. 8a. Aktivität C befindet sich aktuell im Zustand RUNNING, hat also das Datenelement d_1 bereits gelesen. Wird nun die Lesekante von C auf d_1 gelöscht und eine neue Lesekante von C auf d_2 eingefügt – diese Änderung ist auf Schemaebene korrekt – kann es beim Zurücksetzen im Fehlerfall zu Problemen kommen, da unter Umständen falsche Parameterwerte bei eventuellen Rücksetzoperationen herangezogen werden [29]. Bei Zugrundelegung des bisher vorgestellten Verträglichkeitskriteriums wäre der skizzierte Fall jedoch zulässig.

Aus diesem Grund benötigen wir ein angepasstes Verträglichkeitskriterium, das auch Datenflussaspekte einbezieht. Hierzu reichern wir die bisherige Ablaufhistorie für WF-Instanzen um Informationen aus den Datenelementhistorien an. Bezogen auf Abb. 8a könnte sich eine solche erweiterte Ablaufhistorie wie in Abb. 8b darstellen. Aus ihr geht eindeutig hervor, welche Datenelemente eine Aktivität mit welchen Werten gelesen bzw. geschrieben hat. Formal:

Definition 3.4 (Erweiterte Ablaufhistorie). *Seien S ein korrektes WF-Schema mit Kontrollfluss-Schema CFS = (N, E, ...) und DF-Schema DFS und I eine WF-Instanz auf S mit Ablaufhistorie $\mathcal{A}$. Dann heißt $\mathcal{A}_{ext}$ die um Datenelementzugriffe erweiterte Ablaufhistorie von $\mathcal{A}$, wenn $\mathcal{A}_{ext}$ für jeden Starteintrag einer Aktivität die Werte der gelesenen Datenelemente und für jeden Endeintrag entsprechend die Werte geschriebener Datenelemente enthält:*

$$\mathcal{A}_{ext} := < e_1^{(d_1^{(1)},v_1^{(1)}),\ldots,(d_n^{(1)},v_n^{(1)})}, \ldots, e_k^{(d_1^{(k)},v_1^{(k)}),\ldots,(d_m^{(k)},v_m^{(k)})} >$$

Ein Tupel $(d_i^{(\mu)}, v_i^{(\mu)})$ beschreibt dabei einen Lese- bzw. Schreibzugriff von e_μ auf das Datenelement $d_i^{(\mu)}$ (mit zugehörigem Wert $v_i^{(\mu)}$).

Auf Grundlage dieser Definition geben wir nun ein modifiziertes Verträglichkeitskriterium für WF-Instanzen an, das auch in Verbindung mit Änderungen des DF-Schemas anwendbar ist (hier zunächst ohne Berücksichtigung von Schleifen):

Definition 3.5 (Verträglichkeit bei Kontroll- und Datenflussänderungen). *Seien S ein korrektes WF-Schema mit Kontrollfluss-Schema CFS = (N, E, ...) und DF-Schema DFS und I eine WF-Instanz von S mit erweiterter Ablaufhistorie $\mathcal{A}_{ext} = < e_1^{(d_1^{(1)},v_1^{(1)}),\ldots,(d_n^{(1)},v_n^{(1)})}, \ldots, e_k^{(d_1^{(k)},v_1^{(k)}),\ldots,(d_m^{(k)},v_m^{(k)})} >$. S werde durch eine Änderung Δ auf das (korrekte) WF-Schema S' transformiert. Dann:*

I verträglich mit S' :$\Leftrightarrow$ $\mathcal{A}_{ext}$ ist auch auf S' erzeugbar

Zum einen müssen die in Def. 3.1 geforderten Bedingungen für Kontrollflussänderungen weiter erfüllt sein, zum anderen muss zusätzlich gelten, dass jede bereits gestartete Aktivität bei Ausführung auf dem neuen Schema dieselben Datenelemente lesen würde und dass jede beendete Aktivität dieselben Datenelemente geschrieben hätte. Nun stellt sich die Frage, wie die Verträglichkeit einer WF-Instanz bei Änderungen des DF-Schemas überprüft werden kann. Ziel ist wieder, einfache Bedingungen anzugeben, bei deren Einhaltung die Verträglichkeit der betrachteten WF-Instanz mit dem geänderten WF-Schema gewährleistet ist. Tab. 2 fasst zusammen, welche Statusvoraussetzungen für DF-Änderungsoperationen zu fordern sind, um Verträglichkeit im Sinne von Def. 3.5 zusichern zu können. Das Einfügen eines Datenelements ist immer möglich, da im Verlauf der WF-Ausführung noch keine Aktivitäten lesend oder schreibend darauf zugegriffen haben. Das Löschen eines Datenelements ist nur dann unkritisch, wenn noch keine Lese- oder Schreibzugriffe stattgefunden haben. Lesekanten auf Datenelemente können eingefügt oder gelöscht werden, wenn die zugehörige Aktivität noch nicht gestartet wurde. Schreibkanten können sogar noch manipuliert werden, wenn die zugehörige Aktivität bereits gestartet, aber noch nicht beendet wurde.

Wie bereits erwähnt, werden auch beim Einfügen und Löschen von Aktivitäten begleitende Anpassungen des Datenflusses vorgenommen. Hier sind die Voraussetzungen aus Tab. 2 automatisch erfüllt, wenn die Statusbedingungen für die entsprechende Knoteneinfüge- bzw. Knotenlöschoperation gelten (vgl. Tab. 1).

Zu untersuchen bleibt, wie Def. 3.3 (Verträglichkeit bei Schleifen) und Def. 3.5 zusammenspielen. Als Basis für beide Varianten des Verträglichkeitskriteriums dient eine modifizierte Ablaufhistorie $\mathcal{A}_{ext,red}$. Dazu wir $\mathcal{A}$ zunächst gemäß Def.

Tabelle 2. Änderungsoperationen auf dem DF-Schema

	I ist verträglich mit S' $\Leftrightarrow$
Datenelement d wird in das Datenfluss-Schema eingefügt bzw. gelöscht	[Es existiert keine Aktivität mit Status RUNNING oder COMPLETED, die lesend oder schreibend auf dieses Datenelement zugegriffen hat.]
Einfügen/Löschen von	I ist verträglich mit S' $\Leftrightarrow$
Lesekante d $\rightarrow$ n	NS(n) $\in$ {NOT_ACTIVATED, ACTIVATED, SKIPPED}
Schreibkante n $\rightarrow$ d	NS(n) $\neq$ COMPLETED

3.4 reduziert ($\rightarrow \mathcal{A}_{red}$) und dann um die Informationen aus der Datenhistorie angereichert ($\rightarrow \mathcal{A}_{ext}$). Eine WF-Instanz I ist genau dann verträglich mit einem geänderten Schema S', wenn $\mathcal{A}_{ext,red}$ auch auf S' erzeugbar ist.

4 Migration verträglicher WF-Instanzen

Wir gehen nun darauf ein, wie verträgliche WF-Instanzen automatisch und korrekt auf das neue WF-Schema migriert werden können. Dazu sei ein WF-Schema S gegeben, das in ein wiederum korrektes WF-Schema S' abgeändert wird. Ferner seien $I_1, \ldots, I_n$ Instanzen, die auf Grundlage von S erzeugt wurden und die verträglich mit S' sind. Für sie stellt sich nun die Frage, wie ihre Migration auf S' konkret erfolgen soll. Insbesondere muss für jede Instanz I_k geklärt werden, ob ihre bisherigen Zustandsmarkierungen und Arbeitslisteneinträge unverändert übernommen werden können oder ob Anpassungen erforderlich werden. Das betrifft auch den Status von Knoten und Kanten, die infolge der Schemaänderung neu erzeugt wurden.

Abhängig von Art und Umfang einer Schemaänderung sowie aktuellem Status einer zu migrierenden WF-Instanz I_k können die erforderlichen Zustandsanpassungen mehr oder weniger umfangreich ausfallen. Betrifft die Schemaänderung einen Bereich des WF-Graphen, den die Instanz aktuell noch nicht betreten hat, so sind – abgesehen von der Initialisierung neu hinzukommender Knoten und Kanten – keine Zustandsanpassungen erforderlich. Anders verhält es sich, wenn der Kontext einer Änderung auch Knoten und Kanten umfasst, die bereits markiert wurden. So kann es beim Einfügen einer Aktivität (und ihrer Kontextkanten) erforderlich werden, dass die Aktivierung bisher aktivierter (und damit in Arbeitslisten gestellter) Schritte wieder aufgehoben oder umgekehrt die neu hinzugefügte Aktivität sofort aktiviert wird. Ähnliches gilt auch für das Einfügen oder Entfernen von Kontroll- bzw. Sync-Kanten auf Instanzebene. Für komplexe Schemaänderungen, die mehrere Elementaroperationen umfassen, können die notwendigen Markierungsanpassungen sehr viel umfangreicher ausfallen. Hier ist die Beantwortung der Frage, wie Markierungen bei Instanzmigrationen effizient und konsistent angepasst werden können, im Allgemeinen nicht trivial.

Wir geben im Folgenden ein Verfahren an, mit dem sich die Knoten- und Kantenmarkierungen von Instanzen bei Migrationen automatisch anpassen lassen. Dabei werden jeweils nur Knoten und Kanten der von der Änderung betroffenen Bereiche des WF-Graphen neu bewertet, wodurch sich der Gesamtaufwand gegenüber einer Neubewertung aller Knoten und Kanten erheblich reduzieren lässt. Des Weiteren stellt das Verfahren sicher, dass für eine WF-Instanz nach ihrer Migration wieder ein korrekter und konsistenter Zustand resultiert.

4.1 Grundlagen

Grundlegend für das nachfolgende Verfahren sind die bereits erwähnten Ausführungseigenschaften des ADEPT-Metamodells: Zum einen bleiben Markierungen beendeter bzw. nicht mehr ausführbarer Knoten beim Fortschreiten des Kontrollflusses (in Vorwärtsrichtung) erhalten, zum anderen gibt es für die Markierung und Ausführung von Aktivitätenknoten wohl definierte, einfach implementierbare Regeln, vergleichbar den Schaltregeln bei Petri-Netzen [28]. *Markierungsregeln* geben (abhängig vom Knotentyp) an, welche Ausgangskanten bei Beendigung der Aktivität mit TRUE und welche mit FALSE markiert werden sollen. Umgekehrt definieren *Ausführungsregeln* die Bedingungen, die für eingehende Kanten eines Knotens gelten müssen, damit er als ACTIVATED bzw. SKIPPED markiert werden darf (siehe Abb. 9).

4.2 (Initiale) Bestimmung neu zu bewertender Knoten und Kanten

Im Allgemeinen sollten bei einer Migration nur diejenigen Knoten und Kanten einer Neubewertung unterzogen werden, für die potenziell eine Statusanpassung erforderlich wird. Wir zeigen zuerst, wie sich diese eingeschränkte Knoten- bzw. Kantenmenge bestimmen lässt. Im nachfolgenden Abschnitt gehen wir dann darauf ein, wie hiervon ausgehend die Neubewertung der Zustandsmarkierungen bei der Migration einer verträglichen WF-Instanz erfolgen kann.

Ermittelt werden müssen die Menge E_{check} der Kanten und die Menge N_{check} der Knoten, deren Status bei der Migration von Instanzen neu bewertet werden soll. Tabelle 3 gibt an, wie sich diese Mengen bei Anwendung elementarer Änderungen darstellen. So muss beim Einfügen einer Aktivität X (inkl. ihrer Kontextkanten) der Status ihrer direkten Nachfolgerknoten sowie der in X einmündenden Kontrollkanten neu bewertet werden, um inkonsistente Markierungen ausschließen zu können (vgl. Tab. 3, 2. Zeile). (Eine Neubewertung von X selbst wird nur dann erforderlich, wenn eine der in X einmündenden Kanten mit TRUE_SIGNALED bzw. FALSE_SIGNALED markiert werden kann.) Bei Hinzufügen einer Kontroll- oder Sync-Kante $n_{src} \rightarrow n_{dest}$ wiederum müssen sowohl die Kante als auch ihr Zielknoten n_{dest} neu bewertet werden (vgl. Tab. 3, 3. Zeile). Letzteres ist auch dann erforderlich, wenn die Kante selbst zu NOT_SIGNALED

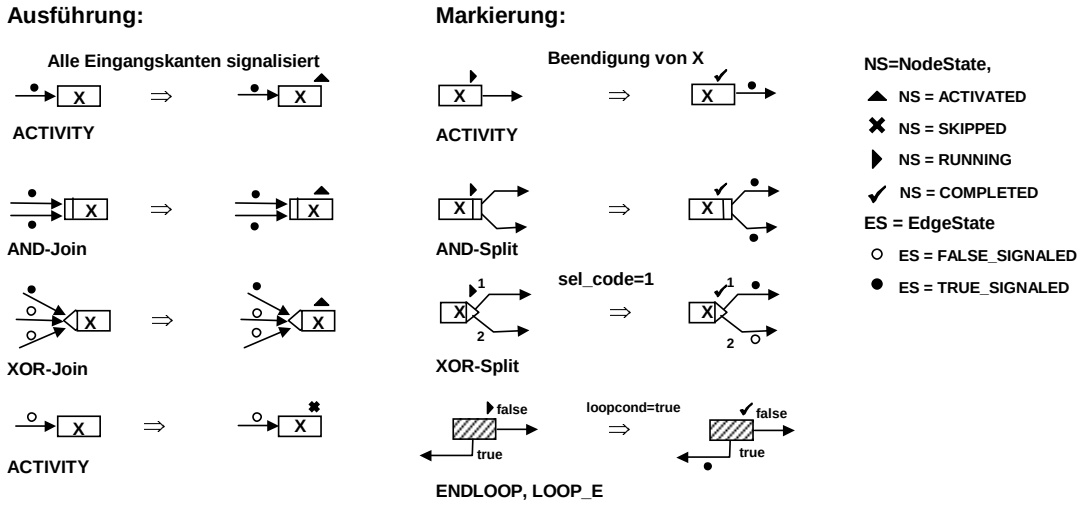


Abb. 9. Ausführungs- und Markierungsregeln (exemplarische Darstellung)

Tabelle 3. Anzupassende Knoten- und Kantenmengen (Auswahl)

Angewandte Operation op...	**... und neu zu bewertende Knoten und Kanten**
Hinzufügen eines Aktivitätenknotens X	N_{check}(op) := {n ∈ N' | X → n ∈ E'}
(inkl. Kontextkanten)	E_{check}(op) := {n_{src} → n_{dest} ∈ E' | n_{dest} = X}
Hinzufügen einer	E_{check}(op) := {n_{src} → n_{dest}}
Kontroll- bzw. Sync-Kante n_{src} → n_{dest}	N_{check}(op) := {n_{dest}}
Löschen einer Kontroll- bzw. Sync-Kante n_{src} → n_{dest}	N_{check}(op) := {n_{dest}}
Löschen eines Aktivitätenknotens X	E_{check}(op) := E_{add}
(inkl. Hinzunahme oder	N_{check}(op) := {n ∈ N | X → n ∈ E}
Entfernen von Kontextkanten)	(E_{add}: Menge der neu hinzukommenden Kanten)
Einfügen eines neuen Teilzweigs Z	E_{check}(op) := {e}
in eine XOR-Verzweigung	(e: Kontextkante *XOR-Splitknoten* → *Z*)
Einfügen eines Schleifenblocks (n_1, n_2) mit	N_{check}(op) := {n ∈ N' | n_2 → n ∈ E'}
Startknoten n_1 und Endknoten n_2	E_{check}(op) := {n_{src} → n_{dest} | n_{dest} = n_1}

Algorithmus 1. (Bestimmung von E_{check} und N_{check} bei komplexen Änderungen)

```
input
    op_1,...,op_n: Zur Anwendung kommende
    Elementaroperationen
output
    E_check, N_check: Menge der neu zu bewertenden
    Kanten/Knoten
begin
    E_check := ∅; N_check := ∅; //Initialisierung
    for i:=1 to n do //Aggregation
        E_check := E_check ∪ E_check(op_i);
        N_check := N_check ∪ N_check(op_i);
    done
    E_check := E_check ∩ E'; N_check := N_check ∩ N;
end
```

evaluiert. Für diesen Fall muss eine möglicherweise bereits erfolgte Aktivierung von n_{dest} aufgrund des Hinzufügens der Kante wieder aufgehoben werden.

Wie lassen sich E_{check} und N_{check} bei Anwendung komplexer Änderungen, die mehrere Operationen $op_1, \ldots, op_n$ umfassen, ermitteln? Die Vermutung liegt nahe, dass sich diese Mengen durch Vereinigung der aus der Anwendung elementarer Operationen resultierenden Mengen $E_{check}(op_i)$ bzw. $N_{check}(op_i)$ bestimmen lassen. Dem ist jedoch nicht immer so, da sich Einzeloperationen bei der Definition der Gesamtschemaänderung aufeinander beziehen können. Insbesondere können durch Anwendung von op_i die Effekte vorangehend definierter Änderungen $op_{i-1}, \ldots, op_1$ teilweise wieder aufgehoben werden. Algorithmus 1 berücksichtigt diesen Aspekt. Hier ergibt sich E_{check} durch Vereinigung der entsprechenden Kantenmengen der angewandten Operationen $op_1, \ldots, op_n$, wobei noch diejenigen Kanten entfernt werden, die im neuen WF-Schema nicht enthalten sind. Sie wurden im Verlauf der Änderungsdefinition zwar erzeugt, sind dann aber infolge der Anwendung weiterer Operationen wieder weggefallen. Was N_{check} anbetrifft, müssen nur Knoten betrachtet werden, die bereits im Quell-Schema S enthalten waren. (Das nachfolgende Verfahren ist so konstruiert, dass neu hinzugekommene Aktivitäten nur bewertet werden, wenn eine einmündende Kante im Verlauf des Verfahrens neu markiert wird.) Algorithmus 1 zieht also nur die unbedingt notwendigen Knoten-/Kantenmengen zur Neubewertung heran, wodurch die für Algorithmus 2 benötigten Eingabegrößen minimal gehalten werden.

Algorithmus 2 (Neubewertung von Zustandsmarkierungen bei Migrationen)

```
input
   NS, ES: Bisherige Knoten-/Kantenmarkierungen von I (bezogen auf S=(N, E))
   E_check, N_check: Menge der neu zu bewertenden Kanten bzw. Knoten
output
   NS', ES': Neu berechnete Knoten-/Kantenmarkierungen von I (bezogen auf S'=(N', E'))
begin
   // Initialisierung von ES'
   forall e ∈ E' do
      if e ∈ E then // Kante war bereits in bisheriger Kantenmenge E enthalten
         ES'(e):= ES(e)
      else // Kante mit NOT_SIGNALED initialisieren
         ES'(e):= NOT_SIGNALED
   done
   // Initialisierung von NS'
   forall n ∈ N' do
      if n ∈ N then // Knoten war bereits in bisheriger Knotenmenge N enthalten
         NS'(n):= NS(n)
      else // Knoten mit NOT_ACTIVATED initialisieren
         NS'(n):= NOT_ACTIVATED
   done
   // Neubewertung der Kanten aus E_check bzw. der Knoten aus N_check
   repeat
      while E_check ≠ ∅ do
         Entnehme eine Kante e = n_src → n_dest aus E_check;
         Überprüfe durch Anwendung der ADEPT-Markierungsregeln auf n_src, ob
         e mit ES'(e) = TRUE_SIGNALED bzw. ES'(e) = FALSE_SIGNALED
         markiert werden darf - falls ja, passe Kantenmarkierung entsprechend an
         if Markierung von e geändert then
            N_check := N_check ∪ {n_dest}
         endif
         done
      while N_check ≠ ∅ do
         Entnehme einen Knoten n aus N_check;
         if NS'(n) ∈ {ACTIVATED, NOT_ACTIVATED} then
            Überprüfe durch Anwendung der ADEPT-Ausführungsregeln auf n,
            ob dieser Knoten mit NS'(n) ∈ {NOT_ACTIVATED, ACTIVATED, SKIPPED}
            markiert werden darf - falls ja, passe die Markierung entsprechend an
            Falls NS'(n) auf SKIPPED gesetzt wurde:
               E_check:= E_check ∪ {e = n_src → n_dest ∈ E' | n_src = n }
         endif
      done
   until E_check = ∅ and N_check = ∅;
end
```

4.3 Verfahren zur Anpassung von Markierungen bei Migrationen

Ein Schema S werde durch Anwendung der Operationen $op_1, \ldots, op_n$ in ein wiederum korrektes Schema S' abgeändert. Ferner seien E_{check} und N_{check} durch Algorithmus 1 bestimmt. Algorithmus 2 (siehe unten) bestimmt dann die neuen Knoten- und Kantenmarkierungen (NS', ES') einer mit S' verträglichen WF-Instanz I von S nach ihrer Migration auf S'. Im Kern basiert er auf den beschriebenen Ausführungs- und Markierungsregeln (vgl. Abb. 9). Ausgangspunkt bilden die bei der Änderungsdefinition ermittelten Mengen N_{check} und E_{check}. Ergeben sich bei der Bewertung von Knoten und Kanten jedoch Anpassungen, werden ggf. auch weitere Knoten und Kanten neu bewertet (z.B. kaskadierende Markierung der Knoten abgewählter Zweige).

Mittels Algorithmus 1 und 2 werden Art und Umfang der erforderlichen Statusanpassungen gegenüber den Alternativen 1 und 2 (vgl. Abschnitt 4.1) erheblich vereinfacht bzw. reduziert. Während Algorithmus 1 nur einmalig bei der Änderungsdefinition anzuwenden ist, muss Algorithmus 2 für jede zu migrierende Instanz ausgeführt werden. Der Aufwand von Algorithmus 2 kann durch $O(n)$ (n ist die Anzahl der Aktivitätenknoten des WF-Schemas) abgeschätzt werden. Hinzu kommt für jede Instanz noch der Aufwand $O(n)$ für die Verträglichkeitsprüfung (ein Vergleich mit anderen Verfahren findet sich in Abschnitt 5). Durch die Konstruktion des Verfahrens ist in der Mehrzahl der Fälle aber nur eine kleine Teilmenge der Knoten und Kanten neu zu bewerten. Dies gilt selbst dann, wenn die Änderungen in verschiedenen Bereichen des WF-Graphen vorgenommen wurden (im Gegensatz etwa zu dem in [13] beschriebenen Ansatz). Da WF-Schemata mehrere Dut-

zend (oder noch mehr) Aktivitäten umfassen können [35], ist eine solche Reduzierung auch zwingend erforderlich.

Schließlich kann man formal zeigen, dass bei Anwendung dieses Verfahrens auf verträgliche Instanzen im Anschluss wieder eine konsistente Markierung resultiert. Dieselbe Markierung erhält man auch, wenn man die komplette Ablaufhistorie auf dem neuen Schema „nachspielt". Dies wäre aber mit einem wesentlich höheren Aufwand verbunden, da dann sehr viel mehr Knoten und Kanten, in Verbindung mit Schleifen gegebenenfalls sogar mehrfach, bewertet werden müssten. Fügt man z.B. in das Schema aus Abb. 7a eine neue Aktivität X zwischen E und G ein, so müssen bei unserem Ansatz lediglich die neu hinzukommende Kontrollkante E → X sowie die Knoten G und X neu bewertet werden, um nach Migration der Instanz wieder eine konsistente Markierung zu erhalten.

Beispiel (Markierungsanpassungen). Wir betrachten eine Schemaänderung, bei der Anordnungsbeziehungen zwischen Knoten modifiziert werden. Im WF-Graphen aus Abb. 10a sollen B und C nun parallel angeordnet werden. Intern wird diese Änderung durch Hinzunahme der Kanten B → D und A → C sowie durch Entfernen von B → C realisiert. Alg. 1 liefert N_{check} = {C, D} und E_{check} = {A → C, B → D}. Gegeben seien nun Instanzen I_1 und I_2 von S (vgl. Abb. 10c). Sowohl I_1 als auch I_2 sind aufgrund ihrer aktuellen Markierungen verträglich mit S' und können deshalb migriert werden. Bei Migration von I_1 auf S' werden die beiden Kanten B → D und A → C gemäß Alg. 2 mit TRUE_SIGNALED markiert. Sowohl für D als auch C erfolgt jedoch keine Anpassung der Knotenmarkierung, da die Bedingungen für die Ausführung dieser Knoten entweder noch nicht erfüllt sind (Knoten D) oder der betreffende Knoten bereits gestartet wurde (Knoten C). Bei Migration von I_2 werden zunächst die Markierungen von B → D und A → C neu bewertet, wobei lediglich A → C mit TRUE_SIGNALED markiert werden kann. Die anschließende Bewertung von C ergibt, dass dieser Knoten mit ACTIVATED markiert und somit in Arbeitslisten gestellt werden kann.

Die Algorithmen 1 und 2 funktionieren in Verbindung mit beliebig komplexen Änderungen. Das trifft auch für Änderungen von Verzweigungen und Schleifen zu, etwa das Einfügen neuer Teilzweige in eine XOR-Verzweigung. Hier kann es auf Instanzebene sogar vorkommen, dass der neu hinzugefügte Zweig nach Neubewertung der Markierungen vollständig „abgewählt" wird (d. h. Knoten mit SKIPPED und Kanten mit FALSE_SIGNALED bewertet sind). Schließlich liefert Algorithmus 2 auch wichtige Informationen zur Anpassung von Arbeitslisten. So müssen für jede Aktivität des alten Schemas, die vor der Migration aktiviert war und deren Aktivierung bei der Neubewertung der Markierungen aufgehoben wird, Arbeitslisteneinträge wieder entfernt werden. Umgekehrt müssen für bisher nicht aktivierte bzw. neu hinzukommende Knoten, die nach Neubewertung den Status ACTIVATED besitzen, Arbeitslisteneinträge generiert werden.

4.4 Umgang mit nicht verträglichen WF-Instanzen

Der Umgang mit nicht verträglichen Instanzen stellt ebenfalls eine wichtige Herausforderung dar. In der Literatur (z.B. [34]) wird hierzu meist vorgeschlagen, nicht verträgliche WF-Instanzen bei Bedarf in ihrer Ausführung soweit zurückzusetzen, dass sie wieder einen mit S' verträglichen Zustand erreichen. Ein solch partielles Zurücksetzen ist auch in ADEPT möglich. Wir haben darüber hinaus weiterführende Strategien entwickelt [33], von denen im Folgenden eine (besonders interessante) Variante im Zusammenhang mit Änderungen auf Schleifen vorgestellt wird. Gegeben sei dazu eine WF-Instanz, die zum Zeitpunkt der Änderungsdefinition nicht verträglich mit dem geänderten Schema S' ist. Wird nun die Schleifenrücksprungkante später mit TRUE_SIGNALED bewertet, erfolgt eine Neubewertung aller Knoten des Schleifenkörpers mit NOT_ACTIVATED. Damit sind die Voraussetzungen für eine Migration von I auf S' (entsprechend Def. 3.3) „verspätet" erfüllt. Frage ist nun, wie mit solchen WF-Instanzen umgegangen werden soll. In keinem Fall darf eine unverträgliche WF-Instanz sofort auf das neue WF-Schema migrieren, da ansonsten die eingangs genannten Probleme resultieren können. Prinzipiell könnten solche WF-Instanzen dauerhaft nach dem alten Schema weiterlaufen, unabhängig davon, ob sie in der Folge wieder in einen verträglichen Zustand gelangen oder nicht. Dies widerspricht jedoch dem Prinzip, dem Benutzer eine möglichst große Menge migrierbarer Instanzen zu bieten. Eine alternative Variante ist die verzögerte Migration von Instanzen (*delayed migration*).

Beispiel (Verzögerte Migration). In Abb. 11 laufen zum Zeitpunkt $t = 1$ drei WF-Instanzen I_1, I_2 und I_3 auf WF-Schema S. Zum Zeitpunkt $t = 2$ findet eine Schemaänderung statt, die S auf S' transformiert und infolge derer die bisherigen Instanzen auf S' migrieren sollen (Migration M_1). I_1 sei zum Zeitpunkt $t = 2$ verträglich mit S' und kann folglich sofort auf S' migriert werden. I_3 sei zum Zeitpunkt $t = 2$ und zu keinem der folgenden Zeitpunkte verträglich mit S' und soll deshalb weiterhin Schema S verwenden. I_2 ist zwar zum Zeitpunkt $t = 2$ nicht verträglich mit S'. Das System erkennt jedoch, dass I_2 sich innerhalb einer Schleifenausführung befindet und damit evtl. verzögert auf S' migriert werden kann. Deshalb wird I_2 als „pending to migrate (M_1) to S' " eingestuft. In diesem Zustand wartet I_2 auf einen potenziellen Schleifenrücksprung. Das entsprechende Ereignis wird intern in einer *ECA-Tabelle* (**E**vent **C**ondition **A**ction) verwaltet. Tritt eine Bewertung der Schleifenrücksprungkante mit TRUE_SIGNALED ein, wird die zugehörige Aktion, nämlich eine Migration von I_2 auf S', ausgelöst.

Ereignisse können UND/ODER-verknüpft sein. So kann eine verzögerte Migration bei verschiedenen Schleifenrücksprüngen stattfinden, wenn sich die Ausführung der WF-Instanz zum Zeitpunkt der Schemaänderung innerhalb einer geschachtelten Schleife befindet. Wird die Rücksprungkante einer inneren Schleife nicht mit TRUE_SIGNALED bewertet, kann die Migration noch bei Signalisierung der Rücksprungkante der äußeren Schleife stattfinden.

Interessant ist der Fall, dass sich eine Instanz I mit Schema S im Zustand „pending" befindet (also auf eine verzögerte Migration auf ein Schema S' wartet) und in dieser Phase eine weitere Schemaänderung S' → S'' stattfindet. Zunächst ist I von dieser Änderung nicht betroffen, da sich I auf einer Schemaversion vor der von der Änderung betroffenen Version befindet. Kommt es jedoch später zur verzögerten Migration von I auf S', muss analysiert werden, ob die letzte Schema-

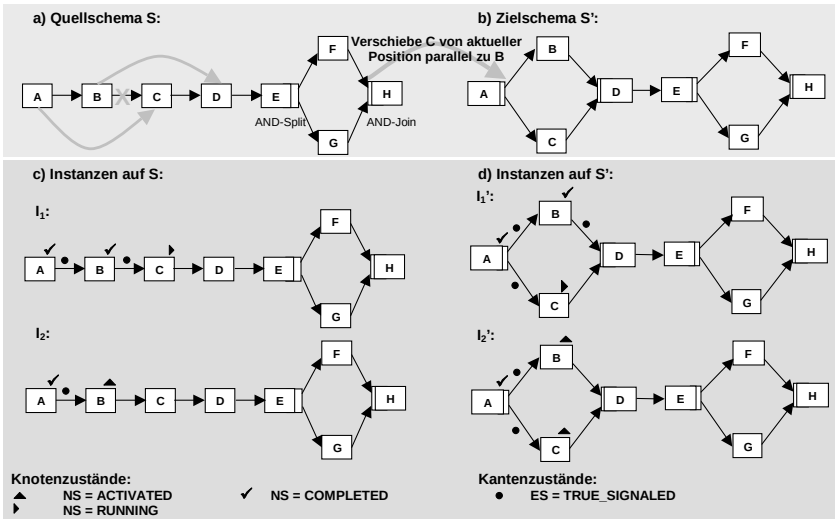
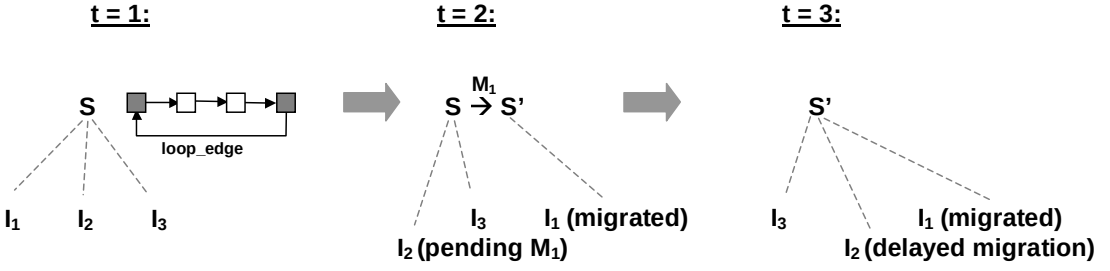


Abb. 10. Ordnungsverändernde Operation auf WF-Schema in ADEPT



Event	Condition	Action
Loop_Back	ES(loop_edge) = TRUE_SIGNALED	migrate I_2 to S'

Abb. 11. Prinzip der verzögerten Migration (*delayed migration*)

änderung ebenfalls auf I propagiert werden kann. Auch hier können wieder die diskutierten Fälle – eine Migration ist sofort, verz¨ogert oder überhaupt nicht möglich – eintreten.

5 Diskussion und Bewertung

Zunächst diskutieren wir in Abschnitt 5.1 generelle Verfahren für Verträglichkeitsanalysen und Instanzmigrationen und vergleichen sie mit unserem Ansatz. Danach gehen wir in Abschnitt 5.2 auf ausgewählte Verfahren im Detail ein.

5.1 Bewertung und Vergleich alternativer Ansätze

Welche Vorgehensweisen sind für Verträglichkeitsanalysen und Migrationen von Instanzen generell denkbar?

Ansatz 1: Verfahren mit Einbeziehung der kompletten Ablaufhistorie. Einige Ansätze (z.B. [9]) verwenden für Verträglichkeitsanalysen und Migrationen die komplette Ablaufhistorie, indem sie versuchen, alle bisherigen Ablaufereignisse zum Start und Ende von Aktivitäten auf dem neuen Schema „nachzuspielen". Diese Vorgehensweise ist prinzipiell unabhängig von den zur Anwendung gekommenen Änderungsoperationen sowie dem zugrundegelegten WF-Metamodell.

Ansatz 2: Verfahren ohne Historiendaten. Bei diesen Verfahren, die z.B. bei Petri-Netzen anzutreffen sind, wird lediglich der aktuelle Zustand (z.B. Token-Belegungen von Stellen) der zu migrierenden Instanzen berücksichtigt. Üblicherweise wird zunächst der von der Änderung betroffene Bereich des Netzes bestimmt und dann die Änderungspropagation auf diejenigen Instanzen eingeschränkt, deren Ausführung sich aktuell nicht in diesem Bereich befindet (Variante 1). Alternativ dazu wird vorgeschlagen, die Migrationen und die dadurch notwendigen Markierungsanpassungen manuell durch Benutzer vornehmen zu lassen (Variante 2). Dazu müssen entweder Anwender für jede Instanz entsprechende Adaptionen durchführen [14] oder der Modellierer gibt Regeln zur Abbildung der Markierungen zwischen Quell- und Zielschema vor [2,3].

Ansatz 3: Verfahren mit Verwendung „verdichteter" Historieninformation. Hierunter fallen Vorschläge wie der in diesem Beitrag entwickelte Ansatz für die systemseitige Durchführung von Verträglichkeitsanalysen und die automatische Migration verträglicher Instanzen. Insbesondere sollen unnötige Zugriffe auf die komplette Historie vermieden werden und konsistente Markierungsanpassungen erfolgen. Konsistent bedeutet, dass sich für eine Instanz I_k nach ihrer Migration auf das neue Schema S′ dieselben Zu-

standsmarkierungen ergeben, die man auch bei vollständiger Ausführung von I auf Grundlage von S' erhalten hätte.

Wir diskutieren nun die Vorteile unserer Vorgehensweise gegenüber den Ansätzen 1 und 2.

Bewertung von Ansatz 1. Ansatz 1 gestaltet sich für Workflows mit vielen Aktivitäteninstanzen sehr aufwendig. Durch das „Nachspielen" von Ablaufereignissen ergibt sich bereits für WF-Instanzen ohne Schleifen jeweils ein Aufwand von $O(n)$ (n ist dabei die Anzahl der Knoten). In Verbindung mit Schleifen, die möglicherweise viele Iterationen durchlaufen, kann der Historienumfang und damit der Rechenaufwand noch größer ausfallen. Sei beispielhaft ein WF-Schema mit 20 Aktivitäten gegeben, wobei 10 Aktivitäten in einer Schleife mit durchschnittlich 5 Iterationen eingebettet sind. Dann ergibt sich für 20 000 laufende WF-Instanzen eine durchschnittliche Ablaufhistoriengröße von ca. 70 MB, wohingegen die benötigte Markierungsinformation auf 0,2 MB nach oben beschränkt ist.[5]

Darüber hinaus ist Ansatz 1 in Bezug auf Änderungen innerhalb von Schleifen zu restriktiv. Aufgrund der Historieneinträge zu bereits beendeten Iterationen wird möglicherweise eine Instanzmigration verboten, obwohl sie in der weiteren Ausführung nicht zu Inkonsistenzen führen würde. Diese Einschränkung kann mit dem von uns vorgestellten Verträglichkeitskriterium auf Schleifen (siehe Def. 3.3) aufgehoben werden. Zu beachten ist auch, dass die Ablaufhistoriendaten aufgrund ihrer Größe üblicherweise nicht im Hauptspeicher gehalten werden, so dass teure Externspeicherzugriffe notwendig werden. Abgesehen davon werden bei Ansatz 1 zahlreiche unnötige Markierungsanpassungen in Bereichen vorgenommen, die von den Änderungen überhaupt nicht betroffen sind. Außerdem kann nicht direkt festgestellt werden, welche Arbeitslisteneinträge anzupassen sind. Ansatz 1 ist aus diesen Gründen – besonders bei großer Instanzzahl – nicht praktikabel.

Bewertung von Ansatz 2. Der komplette Verzicht auf Historien- oder Verlaufsdaten, wie von Ansatz 2 propagiert, führt in der Praxis ebenfalls zu zahlreichen Problemen. Die vorgestellte Variante 1 erweist sich bei genauerer Betrachtung als zu restriktiv, da auch solche Instanzen von einer Änderungspropagation ausgeschlossen werden können, die bei Anwendung unseres Verfahrens problemlos migrierbar sind. Aufgrund der speziellen Eigenschaften vieler Petri-Netze (z.B. keine klare Trennung von Kontroll- und Datenfluss) können zudem Änderungsbereiche in Relation zum Gesamtnetz sehr groß werden, so dass u.U. die Mehrzahl der Instanzen von einer Migration ausgeschlossen wird. Abgesehen davon ist die Bestimmung der von der Änderung betroffenen Bereiche i. A. sehr aufwendig. Bei dem in [1] vorgestellten Ansatz etwa beträgt die Komplexität hierfür $O(n^4 * (n!)^2)$. Zusätzlich muss noch für jede Instanz geprüft werden, ob sie sich in ihrer Ausführung aktuell in diesem Bereich befindet. Hierfür muss die Token-Belegung des Gesamtnetzes untersucht werden, woraus ein Aufwand von $O(n)$ für jede Instanz resultiert. Um dem Problem zu begegnen, dass mit Variante 1 zu viele Instanzen von einer Änderungspropagation ausgeschlossen werden, gibt es Vorschläge aus dem Petri-Netz-Bereich, die Migration für einzelne Instanzen nach Verlassen des Änderungsbereichs nachzuholen. Dies erfordert jedoch, dass nach jedem normalen Schalten einer Transition zusätzlich überprüft werden muss, ob dieser Fall schon eingetreten ist. Damit erhöht sich die Komplexität des Verfahrens auf $O(n^2)$. Variante 2 verfolgt einen gänzlich anderen Ansatz, indem sowohl die Verträglichkeitsanalysen als auch die Markierungsanpassungen auf den Benutzer verlagert werden. Um sicherzustellen, dass es dabei in der Folge nicht zu Verklemmungen kommt, müssten zudem für jede Instanz aufwendige Erreichbarkeitsanalysen (mit exponentieller Komplexität) durchgeführt werden. Dies würde zu erheblichen Verzögerungen bei der Instanz-Ausführung führen. Variante 2 ist deshalb eher von theoretischer Natur und würde in der Praxis so nicht akzeptiert werden.

Bewertung von Ansatz 3. Bei Verwendung verdichteter Historiendaten, wie in dem von uns entwickelten Ansatz 3, beträgt der Aufwand für Verträglichkeitsanalysen schlechtestenfalls $O(n)$. Dieser Fall tritt in ADEPT aber praktisch nie auf. Wählt man z.B. für das Einfügen eines Aktivitätenknotens n_{insert} eine realistische Wahrscheinlichkeitsverteilung für die Anzahl der Nachfolger von n_{insert}, beläuft sich die erwartete Anzahl an Zustandsüberprüfungen auf zwei. Ein wesentlicher Vorteil bei den von uns durchgeführten Verträglichkeitsanalysen ist auch, dass wir gezielt die Semantik der zur Anwendung gekommenen Änderungsoperationen nutzen. Ein weiterer Vorteil liegt darin, dass die benötigten Markierungsdaten aufgrund ihrer geringen Größe in der Mehrzahl der Fälle im Hauptspeicher gehalten werden können. Darüber hinaus können die bei Migrationen notwendigen Zustandsanpassungen automatisch erfolgen. Die Komplexität des vorgestellten Verfahrens beträgt $O(n)$. Wie gezeigt, wird man in der Mehrzahl der Fälle mit wesentlich geringerem Rechenaufwand zum Ziel kommen.

5.2 Diskussion ausgewählter Ansätze

Ein erster Lösungsansatz zur Evolution von WF-Schemata wurde im Bereich der Petri-Netze in [13] präsentiert. Für die WF-Modellierung und -Steuerung werden sog. Flussnetze verwendet, welche zur Laufzeit mehrere WF-Instanzen mit unterscheidbaren Marken kontrollieren. Eine Änderung des Schemas erfolgt formal durch eine Netztransformation, wobei ein markiertes Subnetz durch ein anderes markiertes Subnetz ersetzt wird. Wie in Abschnitt 5.1 (Ansatz 2, Variante 2) diskutiert, erfordert die Überprüfung der Korrektheit des resultierenden Netzes komplexe Erreichbarkeitsanalysen für jede Instanz. Hinzu kommt, dass der Modellierer für jede WF-Instanz die Markierungsanpassungen selbst vornehmen muss [14]. Außerdem bleiben Fragestellungen wie Anpassungen von Datenflüssen oder Arbeitslisten unbeantwortet.

Ein Beispiel für Ansatz 2, Variante 1 aus Abschnitt 5.1 findet sich in [1]. Hierbei kann eine WF-Instanz nicht auf ein mo-

[5] Ein einzelner Historieneintrag einer WF-Instanz umfasst in ADEPT etwa Informationen wie Ereignistyp (Short, 2 Bytes), Aktivitätenbezeichner (Long, 8 Bytes), Bearbeiter (Long, 8 Bytes), Zeitstempel (Long, 8 Bytes), Iterationszähler (Short, 2 Bytes) und Verzweigungsentscheidung (Short, 2 Bytes). Damit ergibt sich pro Historieneintrag eine Größe von 30 Bytes. Dagegen wird zur Speicherung der jeweiligen Knotenmarkierung bei unserem Ansatz nur 1 Byte benötigt.

Tabelle 4. Vergleich verschiedener Ansätze

Bewertungsaspekte	**Ansatz 1** [9]	**Ansatz 2 (Var.1)** [1]	**Ansatz 2 (Var.2)** [14]	**Ansatz 3** ADEPT
Verträglichkeitsanalysen				
- präzise formale Bedingungen	$+$	$+$	$+$	$++$
- Reduzierung d. Analyse-/Anpassungsaufwands				
• durch Ausnutzung der Änderungssemantik	$--$	$-$	$-$	$+$
• durch Verdichtung der Zustandsinformation	$--$	$+$	$\circ$	$++$
Gesamtaufwand für Verträglichkeitsanalysen	$O(n)$	$O(n^4 * (n!)^2)$ $O(n^2)$	$O(e^n)$	$O(n)$ ~ 2 Op.
- Kosten durch Externspeicherzugriffe	$--$	+	+	+
- Autom. Erkennung aller migrierbaren Instanzen	$\circ$	$-$	$--$	$++$
Automatische Migration verträglicher Instanzen (automatische Zustandsanpassungen)	$+$	$\circ$	$--$	$+$ $O(n)$
Praxistauglichkeit	$\circ$	$--$	$--$	$+$

difiziertes Petri-Netz migrieren, solange sie sich in ihrer Ausführung im von der Änderung betroffenen Bereich befindet. Die Anpassung von Netzmarkierungen bei der Propagation von Schemaänderungen wird als schwieriges Problem (dynamic change bug) erkannt [2,3]. Als Lösung wird vorgeschlagen, dass der Modellierer eine Funktion zur Abbildung der Markierungen zwischen Quell- und Zielnetz angibt, die dann auf jede Instanz anwendbar ist [2]. Dieser Ansatz ist aber allein aus kombinatorischen Gründen nicht immer praktikabel. Das gilt besonders für den Fall, dass durch das Netz nicht nur der Kontrollfluss, sondern auch die Datenflüsse zwischen Aktivitäten abgebildet werden. Abschließend sei erwähnt, dass Petri-Netze noch weitere Probleme, wie fehlende Verlaufsmarkierungen, mangelhafte Trennung von Kontroll- und Datenfluss und implizite Schleifen-Modellierung aufweisen.

Einige Ansätze zur Evolution von WF-Schemata verwenden ein graph-basiertes WF-Metamodell. In WIDE [9] werden dem Modellierer Änderungsoperationen angeboten, mit deren Hilfe ein Schema S in ein korrektes Schema S' transformiert werden kann (statischer Fall). Zur Sicherstellung der Korrektheit bei Migration der von S abgeleiteten, noch laufenden Instanzen auf S' (dynamischer Fall) wird erstmals ein Verträglichkeits-Kriterium verwendet. Leider fehlen Angaben dazu, ob bzw. wie sich unter Verwendung der kompletten Ablaufhistorie (siehe Abschnitt 5.1, Ansatz 1) Verträglichkeit konkret feststellen lässt und wie Instanzen nach ihrer Migration auf das geänderte Schema anzupassen sind.

Versionierungskonzepte bilden einen Schwerpunkt in TRAM [24]. Bei Änderung eines WF-Typs wird eine neue Version abgeleitet und als Sohnknoten im Versionsbaum gespeichert. Dann wird versucht, die nach der alten Version ω laufenden Instanzen unter Erhalt des in [9] definierten Verträglichkeits-Kriteriums auf die abgeleitete Version ω' zu migrieren. Dabei denken die Autoren über eine effiziente Überprüfung der zur neuen Version verträglichen Instanzen nach und schlagen die Definition sog. Migrationsbedingungen vor, anhand derer für jede Instanz überprüft werden kann, ob sie zu einem bestimmten Zeitpunkt auf ω' migriert werden kann. Es gibt keine Hinweise zur Verträglichkeit von Instanzen bei Datenflussänderungen oder im Umgang mit Schleifen.

Aktuelle Ergebnisse stammen aus dem Projekt BREEZE [34]. Die Autoren orientieren sich dabei am ADEPT-Metamodell, wobei der Fokus auf der Behandlung nicht verträglicher WF-Instanzen liegt. WF-Instanzen, die nicht verträglich mit dem geänderten WF-Schema S' sind, sollen über teilweises Rollback (dargestellt durch ein spezielles Konstrukt, den sog. Verträglichkeits-Graphen) in einen verträglichen Zustand rückversetzt werden. Dies ist theoretisch zwar sehr interessant, kann aber nicht immer durchgeführt werden, da Aktivitäten in der Praxis oftmals nicht kompensierbar sind. Es wird weder diskutiert, wie WF-Instanzen (effizient) migriert werden können, noch wie Schleifen oder Datenflüsse behandelt werden sollen.

Einen regelbasierten Ansatz bietet ULTRAflow [15]. Besonderes Augenmerk wird hier auf die Modifikation der Implementierung und der Metadaten von Schritten gelegt. Um einen konsistenten Zugriff der einzelnen Instanzen auf die geänderten Spezifikationen zu regeln, werden spezielle Synchronisationsverfahren verwendet. Zwecks Reduzierung der Komplexität klammern die Autoren jedoch z.B. Löschoperationen oder Datenflussaspekte ganz aus.

Objektorientierte Ansätze finden sich in [21,22,36,38]. In MOKASSIN [22,21] werden Änderungen durch Kapselung von Änderungsprimitiven in den WF-Instanzen realisiert, d.h. die Instanzen selbst sind für den Erhalt der Konsistenz bei Änderungen zuständig. Das Verträglichkeits-Kriterium wird als zu restriktiv eingestuft und eine Lösung über ein feingranulares Versionierungskonzept diskutiert. Aussagen zur (effizienten) Überprüfbarkeit der Verträglichkeit von WF-Instanzen fehlen. Abgesehen davon erschweren die angebotenen Konzepte zur Erweiterbarkeit die Propagierung von Schemaänderungen auf laufende Instanzen. In $WASA_2$ wird ebenfalls ein mächtiges Versionierungskonzept angeboten [36, 37]. Auch hier diskutiert der Autor Möglichkeiten zur effizienten Überprüfung der Verträglichkeit, indem er eine Abbildung zwischen dem geänderten Schema und dem Subworkflow, der sich aus dem Status der zu untersuchenden Instanz ergibt, vorschlägt. Hierbei handelt es sich um einen der wenigen Ansätze, zu denen auch Erfahrungen aus der Implementierung der vorgestellten Konzepte vorliegen [38]. Detaillierte Betrachtungen

zu Schleifen oder zur Anpassung von Datenflüssen wurden unseres Wissens noch nicht veröffentlicht.

In [16] werden nur Schemaänderungen ohne Instanzmigrationen betrachtet. Allerdings ist der Ansatz im Hinblick auf Semantikaspekte bei Schemaänderungen interessant. Als WF-Metamodell werden Statecharts verwendet, über deren Äquivalenz die Semantik von modellierten Abläufen nach Änderungen erhalten bleiben soll. Für WfMS, die Ad-hoc--Abweichungen vom vorgeplanten Ablauf unterstützen (wie z.B. ADEPT), eröffnen sich hierdurch interessante Möglichkeiten der automatischen oder semi-automatischen Schema-Adaption.

6 Zusammenfassung und Ausblick

In diesem Beitrag haben wir einen umfassenden und theoretisch fundierten Ansatz für die bei WF-Schemaänderungen notwendigen Verträglichkeitsprüfungen und WF-Instanzmigrationen vorgestellt. Besonderes Augenmerk lag dabei auf Korrektheits- und Konsistenzaspekten, Benutzerfreundlichkeit und Effizienz. Den daraus resultierenden Anforderungen werden wir einerseits durch effiziente Prüfung auf Verträglichkeit mit dem neuen Schema und andererseits durch automatische Anpassung von Zuständen bei der Migration verträglicher Instanzen gerecht. Wir haben teilweise eine formale Darstellung gewählt, um präzise Aussagen darüber treffen zu können, wann eine Instanz auf ein neues Schema migriert werden darf und welche Informationen für entsprechende Überprüfungen benötigt werden. Dies ist für die Implementation eines solchen Konzepts essenziell. Auch für die Migration verträglicher Instanzen haben wir einen Ansatz vorgestellt. Wir haben uns nicht nur auf Kontrollflüsse beschränkt, sondern auch Datenflussaspekte behandelt. Dabei haben wir Querbezüge und Zusammenhänge aufgezeigt. Erstmals werden in dieser Arbeit auch Schleifen in die Untersuchungen zu Verträglichkeitsprüfungen einbezogen, relevante Problemstellungen diskutiert und Lösungsansätze aufgezeigt. Dabei werden Varianten des in diesem Falle zu restriktiven herkömmlichen Verträglichkeitskriteriums vorgestellt. Die gewonnenen Ergebnisse sind nicht nur spezifisch für ADEPT gültig, sondern können auch auf vergleichbare Ausführungsmodelle [26, 34,36,37] übertragen werden.

Insgesamt werden die vorgestellten Konzepte und Verfahren den eingangs genannten Anforderungen gerecht. Wie erwähnt, bestehen aber noch weiter gehende Anforderungen. Ihre adäquate Handhabung und die Entwicklung entsprechender Lösungskonzepte werden Bestandteil unserer zukünftigen Forschungsarbeiten sein. Hierbei soll auch der Frage nachgegangen werden, ob bzw. inwieweit WF-Schemaänderungen auch auf WF-Instanzen propagiert werden können, deren Ausführungsgraph in Folge einer Ad-hoc-Änderung strukturell vom ursprünglichen WF-Schema abweicht [31]. Bisherige Ansätze beziehen Ad-hoc-Änderungen entweder gar nicht ein oder aber sie lassen für diesen Fall die Propagierung von Schemaänderungen nicht mehr zu. Außerdem werden wir Semantik-Aspekte bei Verträglichkeitsprüfungen sowie die Änderung weiterer Bestandteile des WfMS in unsere Untersuchungen einbeziehen. Von besonderem Interesse für uns ist das Zusammenspiel der verschiedenen Konzepte und deren Umsetzung in einem lauffähigen System. Nur so lässt sich überprüfen, ob die betreffenden Konzepte in der Praxis und bei Anwendung auf komplexe Ablaufbeispiele adäquat umsetzbar sind. Die in diesem Beitrag beschriebenen Konzepte werden derzeit prototypisch umgesetzt.

Literatur

1. van der Aalst, W.M.P.: *Exterminating the Dynamic Change Bug: A Concrete Approach to Support Workflow Change.* Information Systems Frontiers 3(3): 297–317 (2001)
2. van der Aalst, W.M.P., Basten, T.: *Inheritance of Workflows: An Approach to Tackling Problems Related to Change.* Theoretical Computer Science 270(1-2): 125–203 (2002)
3. van der Aalst, W.M.P., Jablonski, S.: *Dealing with Workflow Change: Identification of Issues and Solutions.* Int'l Journal of Comp. Systems, Science and Engineering, 15(5): 267–276 (2000)
4. Andany, J., Leonard, M., Palisser, C.: *Management of Schema Evolution in Databases.* Proc. Int'l Conf. on Very Large Databases, Barcelona, Sept. 1991, pp. 161–170
5. Bassil, S., Benyoucef M., Keller R., Kropf P.: *Addressing Dynamism in E-negotiations by Workflow Management Systems.* Proc. 13th Int'l Workshop on Database and Expert Systems Applications (DEXA'2002), Aix-en-Provence, France, September 2002
6. Bauer, T., Dadam, P.: *Efficient Distributed Workflow Management Based on Variable Server Assignments.* Proc. CAiSE '00, Stockholm, Juni 2000, pp. 94–109
7. Bauer, T., Reichert, M., Dadam, P.: *Adaptives und verteiltes Workflow-Management.* Proc. Datenbanksysteme in Büro, Technik und Wissenschaft, Oldenburg, März 2001, pp. 47–66
8. Beuter, T., Dadam, P., Schneider, P.: *The WEP Model: Adequate Workflow-Management for Engineering Processes.* Proc. Europ. Concurrent Engineering Conf. 1998, Erlangen, April 1998
9. Casati, F., Ceri, S., Pernici, B., Pozzi, G.: *Workflow Evolution*. Data and Knowledge Engineering 24(3): 211–238 (1998)
10. Conrad, S.: *Föderierte Datenbanksysteme – Konzepte der Datenintegration.* Springer, 1997
11. Dadam, P.: *Verteilte Datenbanken und Client/Server-Systeme.* Springer, 1996
12. Dadam, P., Reichert, M., Kuhn, K.: *Clinical Workflows - The Killer Application for Process-oriented Information Systems?* Proc. 4th Int'l Conf. on Business Information Systems (BIS '00), Poznan, 2000, pp. 36–59
13. Ellis, C.A., Keddara, K., Rozenberg, G.: *Dynamic Change within Workflow Systems.* Proc. Int'l ACM Conf. on Organizational Comp. Sys. (COOCS '95), Milpitas, CA, Aug. 1995, pp. 10–21
14. Ellis, C.A., Maltzahn, C.: *The Chautauqua Workflow System.* Proc. 30th Int'l Conf. on System Science, Maui, Hawaii, 1997
15. Fent, A., Reiter, H., Freitag, B.: *Design for Change: Evolving Workflow Specifications in ULTRAflow*, Proc. Int'l Conf. on Advanced Information Systems Engineering (CAISE '02), Toronto, May 2002, pp. 516–534
16. Frank, H., Eder, J.: *Equivalence Transformations on Statecharts.* Proc. 12th Int'l Conf. on Software Engineering and Knowledge Engineering (SEKE '00), Chicago, July 2000, pp. 150–158
17. Gartner Group:*Why E-Business Craves Workflow Technology*, Report, T-09-4929, Dec. 1999
18. Hammer, M., Champy, J.: *Reengineering the Corporation.* Harper Collins Publ. 1993
19. Hensinger, C., Reichert, M., Bauer, T., Strzeletz, T., Dadam, P.: *ADEPT$_{workflow}$ – Advanced Workflow Technology for the Efficient Support of Adaptive, Enterprise-wide Processes.* Proc. Software Demonstration Track (EDBT '00), Konstanz, März 2000, pp. 29–30

20. Jablonski, S., Böhm, M., Schulze, W.: *Workflow Management Entwicklung von Anwendungen und Systemen.* dpunkt, 1999
21. Joeris, G.: *Defining Flexible Workflow Execution Behaviors.* Proc. GI-Workshop, Enterprise-wide and Cross-enterprise Workflow-Management (Informatik '99), Oct. 1999, pp. 49–55
22. Joeris, G., Herzog, O.: *Managing Evolving Workflow Specifications.* Proc. Int'l Conf. on Cooperative Information Systems (CoopIS '98), New York City, August 1998, pp. 310-321
23. Kappel, G.: *Reorganizing Object Behavior by Behavior Composition - Coping with Evolving Requirements in Office Systems.* Proc. BTW '91, Kaiserslautern, March 1991, pp. 446–453
24. Kradolfer, M., Geppert, A.: *Dynamic Workflow Schema Evolution Based on Workflow Type Versioning and Workflow Migration.* Proc. CoopIS '99, Edinburgh, Sept. 1999, pp. 104–114
25. Leymann, F., Roller, D.: *Production Workflow.* Prentice Hall, 2000
26. Martschat, U.: *Vergleich und Bewertung von Production Workflow-Management-Systemen.* Diplomarbeit, Universität Ulm, Fakultät für Informatik, 2001
27. Müller, R., Rahm, E.: *Dealing with Logical Failures for Collaborating Workflows.* Proc. Int'l 5th Conf. on Cooperative Information Systems, Eilat, 2000, pp. 210–223
28. Oberweis, A.: *Modellierung und Ausführung von Workflows mit Petri-Netzen.* Teubner, 1996
29. Reichert, M.: *Dynamische Ablaufänderungen in Workflow-Management-Systemen.* Dissertation, Universität Ulm, Fakultät für Informatik, 2000
30. Reichert, M., Bauer, T., Fries, T., Dadam, P.: *Modellierung planbarer Abweichungen in Workflow-Management-Systemen.* Proc. Modellierung '02, Tutzing, März 2002, pp. 183–194
31. Reichert, M., Dadam, P.: *ADEPT$_{flex}$ - Supporting Dynamic Changes of Workflows Without Losing Control.* Journal of Intelligent Information Systems 10(2): 93–129, (1998)
32. Reichert M., Rinderle S.: *Änderungsrechte in adaptiven Workflow-Management-Systemen.* Proc. Sichere Geschäftsprozesse, St. Leon-Rot, September 2002, pp. 30–42
33. Rinderle, S., Reichert, M., Dadam, P.: *Effiziente Verträglichkeitsprüfung und automatische Migration von Workflow-Instanzen bei der Evolution von Workflow-Schemata.* Ulmer Informatik-Berichte, Nr. 2002-01, April 2002, *http://www.informatik.uni-ulm.de/pw/berichte*
34. Sadiq, S., Orlowska, M.: *On Capturing Exceptions in Workflow Process Models.* Proc. 4th Int'l Conf. on Business Information Systems (BIS '00), Poznan, April 2000
35. Schultheiß, B., Meyer, J., Mangold, R., Zemmler, T., Reichert, M.: *Prozessentwurf für den Ablauf einer stationären Chemotherapie.* DBIS-5, Universität Ulm, Nov. 1995
36. Weske, M.: *Flexible Modeling and Execution of Workflow Activities.* Proc. 31st Int'l Conf. on System Sciences, Hawaii, 1998, pp. 713–722
37. Weske, M.: *Adaptive Workflows based on Flexible Assignment of Workflow Schemes and Workflow Instances.* Proc. GI-Workshop, Enterprise-wide and Cross-enterprise Workflow-Management (Informatik '99), Oktober 1999, pp. 42–48
38. Weske, M., Hündling, J., Kuropka, D., Schuschel, H.: *Objektorientierter Entwurf eines flexiblen Workflow-Management-Systems.* Informatik - Forschung und Entwicklung 13(4): 179–195, (1998)
39. Wiedemuth-Catrinescu, U.: *Evolution von Organisationsmodellen in Workflow-Management-Systemen.* Diplomarbeit, Universität Ulm, Fakultät für Informatik, 2002

Stefanie Rinderle studierte Wirtschaftsmathematik in Augsburg. Seit ihrem Diplom im November 2000 ist sie wissenschaftliche Mitarbeiterin der Universität Ulm, Abteilung Datenbanken und Informationssysteme. Ihre Interessen liegen in den Bereichen Workflow-Management und Änderungsmanagement in adaptiven Workflow-Systemen.

Manfred Reichert ist seit Oktober 2002 Juniorprofessor in der Abteilung Datenbanken und Informationssysteme der Universität Ulm. Hier promovierte er im Juli 2000 zum Thema "Dynamische Ablaufänderungen in Workflow-Management-Systemen". Aktuelle Arbeitsgebiete sind unternehmensweite und -übergreifende WF-Anwendungen, EAI und Workflow sowie verschiedene Aspekte von WfMS.

Peter Dadam ist seit 1990 Professor für Informatik an der Universität Ulm und Leiter der Abteilung Datenbanken und Informationssysteme. Davor war er Leiter der Abteilung Advanced Information Management am Wissenschaftlichen Zentrum der IBM in Heidelberg. Seine aktuellen Arbeitsgebiete sind: Verteilte, kooperative Informationssysteme, Workflow-Management, Datenbanktechnologie und deren Anwendungen in anspruchsvollen Anwendungsgebieten.