Institut für Architektur von Anwendungssystemen

Universität Stuttgart
Universitätsstraße 38
D - 70569 Stuttgart

IBM Deutschland Entwicklung
Schönaicher Strasse 220
D - 71032 Böblingen

Studienarbeit Nr. 2012

Anforderungsnahe Realisierung der Laufzeitmodifikationen WS-BPEL basierter Business-Prozesse

Dennis Hohmann

Studiengang:	Informatik
Prüfer:	Prof. Dr. Frank Leymann
Betreuer:	Andreas Schmitz, IBM
begonnen am:	01.06.2005
beendet am:	30.09.2005
CR – Klassifikation:	H.4.1

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

In den letzten Jahrzehnten hat sich die wirtschaftliche Struktur in den Industrienationen zunehmend verändert. Durch stetige Produktivitätssteigerungen und Vergrößerung der Produktionskapazitäten, haben sich immer mehr Absatzmärkte von Anbietermärkten zu Nachfragemärkten entwickelt. Dies bedeutet, dass der Preis und der Erfolg eines Produktes oder einer Dienstleistung maßgeblich durch die Konsumenten bestimmt wird. Der eigentliche Geschäftsprozess, der hinter einem Produkt oder Dienstleistung steht, muss deshalb, genau wie das Produkt, stets weiterentwickelt und verbessert werden, um im Wettbewerb zu bestehen.

Ein Geschäftsprozess stellt dabei die Arbeitsabläufe innerhalb eines Unternehmens dar. Ein Geschäftsprozess kann eine unterstützende Funktion (administrativ) haben, kann aber auch die eigentliche Wertschöpfung des Unternehmens darstellen (operativ). Für operative Prozesse hat sich in den letzten Jahren die Überzeugung, dass der Geschäftsprozess dem eigentlichen Produkt entspricht, mehr und mehr durchgesetzt. Ein Geschäftsprozess umfasst mehrere Aspekte : die Aufbauorganisation des Unternehmens, die beteiligten Menschen und die Ablauforganisation der eigentlichen Arbeitsschritte, sowie deren Material- bzw. Informationsfluss. Diese Aspekte werden in einem Workflow dargestellt.

1.1 Workflow

Workflows können einmal eine allgemeine Beschreibung eines Geschäftsprozesses sein. Hierbei handelt es sich dann um ein Schema eines Geschäftsprozesses. Schemata sind zustandslos. Workflows können allerdings auch laufende Geschäftsprozesse abbilden. In diesem Fall spricht man von einer Workflow-Instanz. Instanzen werden immer von einem Schema abgeleitet und mit Daten versehen, die den Zustand der Instanz abbilden.

Workflows setzen sich zusammen aus Aktivitäten, diese entsprechen Arbeitsschritten innerhalb eines Geschäftsprozesses. Den Aktivitäten werden Zustände zugewiesen. Diese geben an, ob eine Aktivität gerade ausgeführt wird, bereits ausgeführt wurde oder aus bestimmten Gründen nicht mehr ausgeführt werden kann.

Der Kontrollfluss, d.h. die Reihenfolge in der die einzelnen Aktivitäten ausgeführt werden, wird durch *Conditions* bestimmt. Hierbei handelt es sich um Bedingungen, die erfüllt werden müssen, damit eine Aktivität gestartet werden kann, z. B. muss ein Kreditantrag an eine höhere Stelle weitergeleitet werden, sofern der Kreditbetrag einen bestimmten Wert überschreitet. Analog dazu wird der Datenfluss modelliert. Die Ein- und Ausgabedaten der Aktivitäten werden in Variablen geschrieben und von dort auch wieder ausgelesen.

Workflows können teilweise automatisch ablaufen. Sofern Menschen involviert sind, werden aus den Aktivitäten, die zur Ausführung bereit sind, sogenannte *Workitems* generiert. Ein *Workitem* besteht aus einer Aktivität und einer Anwendung, mit der sie ausgeführt werden soll. Die *Workitems* werden dann in einer *Worklist* dem Anwender präsentiert. Eine *Worklist* ist eine Liste, in der alle Workitems aufgeführt werden, die im Moment dem Nutzer zugewiesen sind. Der Anwender kann nun ein *Workitem* aus der Worklist auswählen. Es wird daraufhin aus den Worklists der andern Anwender entfernt und erhält eine Sperre, so dass es von keinem anderen Anwender mehr ausgewählt werden kann.

1.2 Klassifikation von Workflows

In der Literatur werden im wesentlichen drei Arten von Workflows unterschieden : *Production*, *Ad hoc* und *Administrative* [14, 15, 22, 26]. Diese unterscheiden sich hinsichtlich ihrer Wiederholungshäufigkeit, Vorhersagbarkeit, ihrer strategischen Bedeutung, sowie hinsichtlich des Geldwertes und ihrer Zeitkritizität. In der Literatur wird gelegentlich noch *collaborative* als vierte Klasse genannt [14, 26].

Production Workflows sind im Vorfeld klar definiert und stellen häufig wiederkehrende und gleichartig ablaufende Geschäftprozesse dar. Sie laufen meist in großer Anzahl parallel. Sie stellen außerdem die hauptsächliche Wertschöpfung eines Unternehmens dar. Beipielsweise eine Bestellung. Der Workflow beginnt mit der Bestellung des Kunden und endet mit der Rechnungstellung.

Administrative Workflows sind ebenfalls im Vorfeld klar definiert. Sie zeichnen sich aber nur durch eine geringe strategische Bedeutung, sowie einen geringen Geldwert aus. Es werden hiermit innerbetriebliche Abläufe abgebildet, welche eine unterstützende Funktion haben. Beispielsweise können hiermit die Urlaubsanträge der Angestellten abgewickelt werden.

Ad Hoc Workflows sind nicht vorab definiert und werden in der Regel auch nur einmalig ausgeführt. Der Ablauf der Arbeitsschritte wird erst zur Laufzeit dynamisch ermittelt. In der Praxis sind derartige Systeme häufig eMail-basiert. Die benötigten Dokumente und eine Beschreibung der Aufgabe, werden von den Mitarbeitern per eMail an den zuständigen Sachbearbeiter weitergeleitet.

Collaborative Workflows zielen vor allem auf Teamarbeit ab. Das Endergebnis soll gemeinsam innerhalb einer Gruppe erarbeitet werden [14, 26, 27].

1.3 Workflow Management Systeme

Ein Workflow Management System (WFMS) sorgt dafür, dass der richtige Mitarbeiter die richtigen Arbeitsmittel bzw. Informationen zum richtigen Zeitpunkt, während des Ablaufs eines Geschäftsprozesses, erhält. Mittels WFMS soll es den Unternehmen ermöglicht werden, die Prozesskosten, sowie die Durchlaufzeiten der Prozesse zu reduzieren und die Produktivität zu steigern.
Dies wird ermöglicht, indem die Workflows, innerhalb eines WFMS modelliert, kontrolliert und überwacht werden. Bei der Modellierung kann die Anwendungslogik von der Prozesslogik getrennt werden. Das heißt die einzelnen Arbeitsschritte können unabhängig von der tatsächlichen Implementierung durch einen Benutzer, mittels eines graphischen Tools, arrangiert werden. Bisher war die Ablauflogik im Quellcode der Anwendungen versteckt. Dadurch war es sehr viel schwerer die Arbeitsabläufe zu modellieren und gegebenenfalls zu ändern. Durch WFMS wird es nun auch dem IT-Laien ermöglicht, Workflows zu modellieren. Die einzelnen Aktivitäten werden dann bei der Ausführung mit assoziierten Anwendungen ausgeführt. Ferner wird es ermöglicht, Aktivitäten generisch an Rollen, anstatt wie bisher an feste Agenten, zuzuweisen. Dadurch wird der Grad der Flexibilität deutlich erhöht.

Außerdem werden sämtliche Arbeitsschritte in einer Datenbank dokumentiert, in sogenannten *Audit Trails*. Auf diese Weise können die Geschäftsprozesse im Nachhinein analysiert werden. Dadurch können diejenigen Aktivitäten entdeckt werden, welche Engpässe darstellen und die Performanz des gesamten Worflows signifikant herabsetzten. Es können somit enorme Potentiale zur Prozess-optimierung aufgedeckt werden.

Außerdem wird durch ein WFMS *Process Mining* ermöglicht. Viele Geschäftsprozesse sind in den Unternehmen historisch gewachsen und das Management hat häufig keinen detailierten Plan wie diese nun im Detail ablaufen. Der grobe Ablauf ist zwar bekannt, aber welche Bereiche im Unternehmen genau und wie lange passiert werden ist unbekannt. Mittels *Process Mining* können, aus der Datenbank, die tatsächlich ablaufenden Geschäftprozesse im Unternehmen rekonstruiert werden.

Ein WFMS ermöglicht es außerdem, dass ein modifizierter Workflow zuerst nur simuliert wird. Die Eingabedaten können hierbei auf den gespeicherten Werten

aus der Datenbank aufbauen. Auf diese Weise können Schwächen bereits erkannt werden, bevor der Workflow tatsächlich eingesetzt wird.

Überdies bieten WFMS auch die Möglichkeit Kompensationen durchzuführen, sofern dies von der Beschreibungssprache unterstützt wird. Bei der Sprache WS-BPEL [28] beispielsweise werden den einzelnen Aktivitäten jeweils Kompensationsaktivitäten zugewiesen. Semantisch voneinander abhängige Aktivitäten werden daraufhin in Kompensationssphären zusammengefasst. Wenn nun eine Aktivität innerhalb einer Sphäre fehlschlägt, werden alle Aktivitäten, die sich in der gleichen Sphäre befinden, ebenfalls kompensiert. Es ist aber auch ein alternativer Ansatz denkbar, beispielsweise können alle Aktivitäten in einem Transaktionskontext durchgeführt werden und bei einem Fehler ein Rollback durchgeführt werden [25,26].

1.4 Notwendigkeiten für Ad hoc-Modifikationen an Workflows

Häufig ist es erforderlich einen Workflow, während der Laufzeit, zu modifizieren. Hiebei spricht man von einer Ad hoc-Modifikation. Die Modifikationen können auf zwei Ebenen stattfinden. Zum einen auf der Schema-, zum anderen auf der Instanzebene. Die Gründe hierfür sind vielfältig. Ein spezieller Kundenwunsch kann es erforderlich machen, dass ein Workflow angepasst werden muss. In der Regel reicht es in diesem Fall aus, eine einzige laufende Instanz zu modifizieren. Fehler im Prozessablauf oder eine Ausnahmesituation resultieren ebenfalls häufig in Instanzänderungen. Änderungen der rechtlichen Rahmenbedingungen oder Prozessoptimierungen hingegen resultieren häufig in Schemaänderungen. Hierbei wird das Schema eines Prozesses direkt modifiziert. Durch diese Art von Änderung entsteht ein neues Schema, welches als *Variante* oder *Version* des Prozesses bezeichnet wird. In dieser Studienarbeit wird der Begriff Version verwendet werden. Nach einer solchen Änderungen liegen entweder verschiedene aktive Versionen vor oder die laufenden Instanzen, die nach der alten Version gestartet wurden, müssen auf die neue migriert werden. Manche Bereiche, wie z. B. Geschäftsprozesse in Kliniken, sind in der Regel von vielen Änderungen und unerwarteten Ereignissen betroffen, so dass häufige Änderungen an laufenden Instanzen unvermeidlich sind [6, 24]. Ein vollständig ad hoc-basierter Ansatz würde die Vorteile eines WFMS nicht ausschöpfen. Da hier kein optimiertes globales Schema eingesetzt würde. Ein reiner production-basierter Ansatz wäre zu aufwändig, da alle Ausnahmen bereits im Schema berücksichtigt werden müssen. Dadurch würde der Workflow sehr komplex und unhandlich werden. Zudem gibt es auch eine Menge von Ereignissen, die unvorhersehbar sind, und sich deshalb nicht vorab berücksichtigen und modellieren lassen. Es muss deshalb möglich sein, Ad hoc-Änderungen an laufenden Instanzen vornehmen zu können.

Wenn die Kunden die o.g. Möglichkeit für Ad hoc-Modifikationen angeboten bekommen, dann können sie schneller auf Marktänderungen reagieren. Dadurch könnten sie sich einen erheblichen Wettbewerbsvorteil vor ihren Mitbewerbern sichern. Außerdem wird es ermöglicht, spezielle Kundenwünsche individuell zu

bearbeiten. Auf diese Weise kann die Kundenzufriedenheit erheblich gesteigert werden und der Kunde langfristig an das Unternehmen gebunden werden. Außerdem ist der Einsatz von Workflows in hochgradig dynamischen Umfeldern, wie z. B. in Kliniken, nur möglich, wenn sie flexibel an Ausnahmesituationen angepasst werden können [6, 24].

In dieser Studienarbeit sollen nun die Probleme, welche bei Ad hoc-Modifikationen von Prozessen entstehen, aufgeführt werden. Hierbei werden die Probleme betrachtet, welche die Prozesssteuerung betreffen, z. B. Deadlocks oder fehlende Daten. Es werden dabei die Lösungsansätze von unterschiedlichen Forschergruppen in ihren Grundzügen vorgestellt und diskutiert. Die Formalismen all dieser Ansätze zu beschreiben, würde den Rahmen dieser Studienarbeit sprengen, deshalb wird auf Formalismen weitgehend verzichtet. Der Fokus liegt auf den Lösungsideen, welche wiedergegeben und bewertet werden sollen. Vor allem ihre Tauglichkeit in der geschäftlichen Praxis wird betrachtet. Für eine formale Definition dieser Ansätze wird auf die Quellen verwiesen. Nicht eingegangen wird auf Probleme, die eine untergeordnete Schicht betreffen, z. B. Austauschen einer Applikation, die mit einer bestimmten Aufgabe verknüpft ist oder einer externen Datenbank. Auch Änderungen bezüglich den Agenten, die einer Aufgabe zugewiesen werden, sowie Änderungen bezüglich der Aufbauorganisation des Unternehmens und den resultierenden Auswirkungen auf Workflows, werden im Rahmen dieser Studienarbeit nicht berücksichtigt.

2. Strategien für den Umgang mit Änderungen

In der Literatur werden unterschiedliche Verfahren genannt, wie mit Laufzeitmodifikationen von Workflows umgegangen werden kann. Im Wesentlichen lassen sich folgende Kernpunkte herausarbeiten.

2.1 Flush / Versionierung

Das Flush-Verfahren wird angewandt, wenn Modifikationen am Schema vorgenommen werden. Durch eine Modifikation am Schema wird eine neue Version dieses Schemas erzeugt. Sofern es noch laufende Instanzen gibt, welche von dem ursprünglichen Schema abgeleitet wurden, dürfen nun keine Instanzen nach der neuen Version des Schemas gestartet werden. So lange bis alle laufenden Instanzen terminieren [12, 14, 16]. Dieses Verfahren ist nicht sonderlich kompliziert und benötigt keine größeren Änderungen an bestehenden Workflow Management Systemen und kann bei Szenarios eingesetzt werden, bei denen unterschiedliche Prozessdefinitionen nicht gleichzeitig existieren dürfen. Dieses Verfahren wäre denkbar in einem Umfeld, indem nur automatisch und schnell ablaufende Micro-Flows eingesetzt werden. Allerdings ist dieses Verfahren in der Regel für die Praxis nicht geeignet, da bei lang laufenden Prozessen für einen sehr langen Zeitraum keine neuen Instanzen mehr gestartet werden können.

Eine Modifikation dieses Verfahrens, dass diese Probleme umgeht, stellt die Versionierung dar [23, S.7; 18 S.4]. In manchen Quellen wird dieses Verfahren als eine gesonderte Richtlinie betrachtet, aber aus meiner Sicht stellt es lediglich eine Abwandlung des Flush-Konzeptes dar. Bei dem Versionierungsansatz wird jedem geänderten Schema eine eindeutige Versionsnummer zugewiesen. Die Instanzen werden mit den unterschiedlichen Versionen eindeutig assoziiert. Bei sehr lang laufenden Geschäftsprozessen, z. B. einem Leasing-Vertrag, der sich ohne weiteres über mehrere Jahrzehnte erstrecken kann, ist dieses Verfahren

ebenfalls nicht sehr praktikabel, da hier unter Umständen sehr viele unterschiedliche Versionen eines Geschäftsprozesses aktiv sein können, was die Übersichtlichkeit und die Kontrolle erheblich erschwert. Außerdem können einmal gestartete Instanzen nicht mehr von Verbesserungen am zugrunde liegenden Schema profitieren.

2.2 Abort

Dieses Verfahren kann bei Änderungen auf Schemaebene angewandt werden. Sämtliche laufende Instanzen werden einfach abgebrochen und nach dem neuen Schema erneut gestartet [12, 14, 16, 23]. Dieses Verfahren kann vor allem bei radikalen Änderungen in den betrieblichen Abläufen eingesetzt werden, z. B. bei einer Pleite einer Tochterfirma. Dieses Verfahren ist relativ einfach zu implementieren, ist jedoch nur bei solchen WFMS einsetzbar, welche entsprechende Konzepte zur Kompensation von Aktivitäten anbieten, ansonsten könnte dies zu Inkonsistenzen führen. Da viele Aktivitäten Spuren in den betrieblichen Datenbeständen hinterlassen. Die Daten müssen in den Zustand vor der Durchführung der Aktivitäten zurückversetzt werden, bevor neue Instanzen gestartet werden können. Als Beispiel könnte ein Versandhandel dienen. Jede laufende Bestellung reduziert den aktuellen Warenbestand. Wenn nun alle laufenden Bestellungen per Abort abgebrochen würden, müsste der Warenbestand auf den ursprünglichen Wert zurückgesetzt werden. Ansonsten könnten manche Bestellungen fehlerhaft als nicht vorrätig eingestuft werden. Allerdings ist es in der Realität manchmal nicht praktikabel oder gar nicht möglich, laufende Geschäftsprozesse unmittelbar abzubrechen. Auf jeden Fall führt ein Abbruch eines laufenden Geschäftsprozesses zu einem Verlust der bisherigen Arbeit, was für ein Unternehmen nicht wünschenswert ist.

2.3 Adapt

Bei diesem Verfahren werden einzelne Instanzen modifiziert. Das zugrunde liegende Schema bliebt unangetastet. Gründe hierfür können Fehler oder ein geändertes Umfeld sein, so dass einige Instanzen gesondert behandelt werden müssen [12, 14, 16]. Bei Änderungen auf Instanzebene muss generell überprüft werden, ob die Änderungen die Instanz von einem logisch konsistenten Zustand in einen anderen überführen. Dazu muss der aktuelle Zustand der Instanz geprüft werden und ob die gewünschte Änderungsoperation überhaupt anwendbar ist, d.h. das der Workflow nach der Änderung strukturell immer noch korrekt ist.

Ein weiterer wichtiger Punkt ist auch, ob diese Änderungen bereits im Vorfeld modelliert werden sollen. Wenn mögliche Ausnahmen bereits bei der Beschreibung des Workflows berücksichtigt werden, kann dies den Nutzer entlasten. Der Nutzer muss sich dann, sofern der Ausnahmefall eintritt, keine weiteren Gedanken machen, wie er zu behandeln ist. Das heißt auf Änderungsoperationen und anschließende Konsistenzprüfungen, während der

Laufzeit, kann verzichtet werden. Allerdings werden dadurch die Workflow-Beschreibungen im Allgemeinen sehr komplex und unübersichtlich. Überdies sind nicht alle Änderungen bei der Modellierung vorhersehbar. Dieses Vorgehen sollte vermieden werden, da die Vorteile eines WFMS durch komplexere Prozessmodelle wieder aufgehoben würden. Es müssen also Verfahren zum Einsatz kommen, die Modifikationen zur Laufzeit ermöglichen.

Außerdem muss noch geklärt werden, ob die Änderungen temporär oder permanent sein sollen. Das System sollte in der Lage sein, beide Arten von Änderungen auf Instanzebene zu unterstützen, um in der geschäftliche Praxis erfolgreich zu sein und akzeptiert zu werden [13, S12]. Unter temporären Änderungen versteht man Änderungen die nur einmal durchgeführt werden sollen, z. B. nur bei dem aktuellen Schleifendurchlauf. Beim nächsten Schleifendurchlauf oder wenn der Geschäftprozess teilweise rückgängig gemacht wird, soll die temporär eingefügte Änderung nicht mehr vorhanden sein [24, 10 S.22]. Das Gegenstück hierzu stellen permanente Änderungen dar. Hier muss die Änderung bis zum Ende des Workflows erhalten bleiben. Das heißt, im Falle einer Schleife, für alle folgenden Durchläufe oder bei einem Abbruch und Neustart der Instanz muss die Änderung immer noch vorhanden sein. Es stellt sich nun die Frage wie diese zwei Arten von Änderungen dargestellt werden können und wie mit ihnen umgegangen werden soll. Daher sind umfangreiche Prüfungen nötig, um eventuelle Inkonsistenzen zu vermeiden. Dies kann z. B. vorkommen wenn eine permanente Änderung semantisch auf einer temporären Änderung aufbaut. Der Workflow ist dann im Moment korrekt, wenn es nun aber erforderlich wird den Verlauf teilweise zurückzusetzen, kann es vorkommen, dass die temporäre Änderung ungeschehen gemacht wird und die permanente Änderung erhalten bleibt. Dies würde in einer inkonsistenten Struktur des Workflows resultieren. Es wurden bereits Ansätze erarbeitet wie mit solchen Änderungen verfahren werden soll [7, 10, 24]. Die Autoren dieser Papers schlagen vor, zwei Änderungslogs vorzuhalten. In einem sind alle Änderungen aufgeführt, wohingegen im anderen nur die permanenten eingetragen werden. Die Basis für die Ausführung des aktuellen Workflows wird anhand aller Änderungen bestimmt. Bei temporären Änderungen müssen diese zusätzlich noch auf ihre Verträglichkeit mit den permanenten Änderungen überprüft werden [7, S.17].

Meiner Meinung nach eignet sich dieser Ansatz für die Klasse der Production Workflows, da diese in der Regel immer nach dem gleichen Muster ablaufen. Allerdings eignet sich dieser Ansatz nur für kleine Abweichungen von dem vorgegebenen Muster. Komplexere Änderungen können zum einen den Anwender überfordern, zum anderen werden die notwendigen Konsistenzprüfungen dann zu umfangreich.

2.4 Build

Der Aufbau neuer Prozesse wird ebenfalls als eine Art der Modifikation gesehen [12, 14, 16]. Es wird vorgeschlagen dieses Verfahren für Geschäftsprozesse zu verwenden, die nicht vollständig modelliert werden können. Dies kann vorkommen, wenn der Geschäftsprozess in einem Umfeld abläuft, das von vielen Ausnahmen und individuelle Vorgehensweisen geprägt ist. Zum Beispiel wenn die notwendigen Aktivitäten identifiziert werden können, die Reihenfolge aber im Allgemeinen individuell bestimmt werden muss [15]. In Kapitel 8 wird hierzu ein Beispiel präsentiert und näher auf diese Thematik eingegangen.

Dieser Ansatz eignet sich meiner Meinung nach für Geschäftprozesse, welche selten vorkommen und der Ablauf in der Regel nicht vorhersehbar ist. Innerhalb eines Unternehmens sind solche Abläufe vor allem auf der strategischen Ebene anzutreffen. Je nach aktueller Marktlage werden unterschiedliche Informationen benötigt und es sind unterschiedliche Aktivitäten erforderlich. Die einzelnen Aktionen sind zwar im Vorfeld bekannt, aber die Reihenfolge wird erst zur Ausführung ersichtlich. Dieses Verfahren kann also auch für Geschäftsprozesse verwendet werden, denen kein vollständig definiertes Schema zugrunde liegt.

2.5 Migrate

Dieses Verfahren wird ebenfalls bei Änderungen auf der Schemaebene eingesetzt [2, 3, 7, 16, 18] . Gelegentlich wird für dieses Konzept auch der Begriff *Dynamic Change* verwendet [23]. Die Nachteile des Abort- oder Flush-Konzeptes sollen hier gänzlich vermieden werden. Ziel ist es, die laufenden Instanzen auf das neue Schema zu migrieren, so dass sie nach dem geänderten Schema zu Ende laufen.

Dabei muss berücksichtigt werden, dass sich die laufenden Instanzen in unterschiedlichen Zuständen befinden. Da sie in der Bearbeitung unterschiedlich weit fortgeschritten sein können. Deshalb kann es bei einer Migration auf ein geändertes Schema zu Inkonsistenzen kommen. Beispielsweise kann es vorkommen, dass eine Instanz aufgrund ihres Fortschrittes nicht nach dem neuen Schema korrekt terminieren kann. Aus diesem Grund kann es manchmal erforderlich sein, bereits abgeschlossene Aktivitäten rückgängig zu machen, um einen kompatiblen Zustand zu erreichen. Es wurden auch Ansätze gemacht, die Migration weitgehend zu Automatisieren und dadurch Kosten zu sparen [3].

Der Vorteil dieses Konzeptes ist, dass es jeweils nur eine aktive Version eines Prozesses gibt. Überdies können bereits gestartete Instanzen unmittelbar von Prozessoptimierungen profitieren. Für lang laufenden Geschäftsprozesse ist dieser Ansatz aus meiner Sicht den anderen vorzuziehen. Dieses Konzept benötigt aber größere Änderungen an bestehenden WFMS und ist mit einigen Problemen verbunden. Es wurde auch schon viel Forschung in diese Richtung unternommen.

Dabei wurden schon ausgereifte Konzepte und Verfahren entwickelt. Worauf im nächsten Kapitel näher eingegangen wird. Hier werden auch Lösungsansätze, für die entstehenden Probleme, vorgestellt und diskutiert.

3. Probleme bei Migrationen von Instanzen auf geänderte Schemata

Wenn ein Schema geändert wurde, ist es im Allgemeinen wünschenswert auch die bereits gestartet Instanzen auf das neue Schema zu migrieren. Dabei treten aber eine Reihe von Schwierigkeiten auf.

Am Beispiel von ADEPT sollen hier nun diese Probleme aufgezeigt werden und Ansätze zur Lösung vorgestellt und diskutiert werden.

3.1 Grundlagen von ADEPT

Es sollen hier nur kurz die grundlegenden Elemente von ADEPT erläutert werden. Für weitere Informationen wird auf die Quellen verwiesen [2, 7]. ADEPT ist eine bockorientierte Sprache, welche symmetrische Kontrollstrukturen verwendet, d.h. Kontrollflussstrukturen, wie Schleifen oder Verzweigungen, werden durch eindeutige Start- und Endknoten eingegrenzt. Einzelne Blöcke dürfen sich nicht überlappen. In dieser Hinsicht gibt es einige Ähnlichkeiten zur Sprache WS-BPEL [28].

ADEPT bietet zur Modellierung von Workflows unterschiedliche Konstrukte an. Aktivitäten werden mittels Knoten dargestellt. Den Aktivitäten werden Zustände zugeordnet. Zustände können sein : *not activated*, *activated*, *running*, *skipped*, *completed*.[1] Die meisten Begriffe sind selbsterklärend. Zur Unterscheidung zwischen *activated* und *running* muss gesagt werden, dass eine Aktivität im Zustand *activated* bereit zur Ausführung ist. Wenn Menschen involviert sind, werden aus den Aktivitäten im Zustand *activated* Workitems generiert. Wenn die Aktivität dann tatsächlich ausgeführt wird, geht sie in den Zustand *running* über.

[1] Diese Aufzählung ist nicht vollständig. ADEPT bietet zusätzlich Zustände, um übersprungene (skipped) und fehlgeschlagene Aktivitäten (terminated) darzustellen, diese sind aber für das weitere Verständnis nicht notwendig.

Die Kontrollflusslogik wird ebenfalls mittels unterschiedlicher Knoten dargestellt. Hierbei wird ein blockorientierter Ansatz benutzt. Schleifen werden durch einen eindeutigen Schleifenstart- und einen Schleifenendknoten modelliert. Außerdem gibt es noch Split- und Join-Elemente. Ein AND-Split beispielsweise bedeutet, dass ausgehende Pfade gleichermaßen aktiviert werden. Ein XOR-Split, dass nicht alle Pfade weiterverfolgt werden. Welche Pfade letztendlich verfolgt werden, hängt von bestimmten Selektionen ab. Das entsprechende Gegenstück zu Splits sind Joins. Hierbei wird ebenfalls zwischen AND- und XOR-Joins unterschieden.

Der Datenfluss wird mittels globaler Datenelemente modelliert. Die Aktivitäten verfügen über Eingabeparameter, die über Lese- und Schreibkanten auf die Datenelemente zugreifen. Wichtig ist auch, dass bei ADEPT Datenelemente nicht überschrieben werden, sondern dass für jedes Datenelement eine Tabelle angelegt wird, in der die zugewiesenen Werte nacheinander abgelegt werden. Auf diese Weise ist es möglich, im Falle eines Rücksprunges, die ursprünglichen Werte in den Datenfeldern zu rekonstruieren. Ein weiteres wichtiges Element sind Synchronisationskanten mit denen eine „Wartet-auf“-Beziehung zwischen Aktivitäten modelliert werden kann [2, 7].

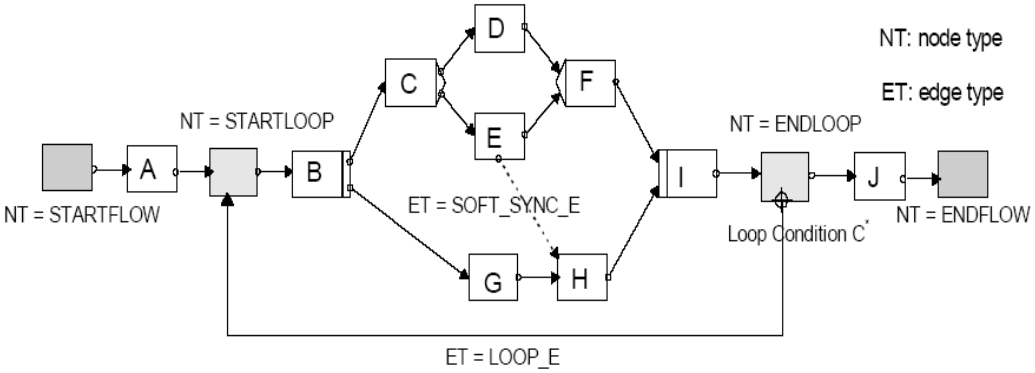

Abbildung 1 : Ein einfaches ADEPT-Beispiel.
Quelle : Reichert, Dadam. *ADEPTflex – Supporting Dynamic Changes of Workflows Without Loosing Control* [7].

In der obigen Abbildung ist ein Beispielworkflow in ADEPT dargestellt. Nach Aktivität B findet ein AND-Split statt, d.h. nachdem B ausgeführt wurde, werden die Aktivitäten C und G aktiviert. Die Aktivität I ist mit dem korrespondierenden AND-Join versehen. Beide einlaufenden Pfade müssen abgeschlossen sein, bevor die Aktivität I gestartet wird. Nach C findet ein XOR-Split statt. In der tatsächlichen Ausführung wird dieser mit einem *selection code* versehen. Dies stellt ein Auswahlkriterium dar, welches bestimmt welcher der ausgehenden Pfade weiterverfolgt wird. Von der Aktivität E geht eine Synchronisationskante nach H aus, d.h. E kann erst aktiviert werden, wenn H aktiviert wurde.

Datenelemente sind in der Abbildung keine dargestellt.

3.2 Vorschläge zu Konsistenzsicherung bei Ad hoc Änderungen

Dadam und Reichert machen einige sehr vielversprechende Vorschläge, wie die Verträglichkeit von laufenden Instanzen mit geänderten Schemata effizient überprüft werden kann. Im Rahmen dieser Studienarbeit werden die Ideen, die hinter diesen Ansätzen stehen, grob wiedergegeben und diskutiert. Für detailiertere Informationen wird auf die Quellen verwiesen.

In Adept ist eine Migration dadurch realisiert, dass die Änderungen, die auf Schemaebene durchgeführt wurden, ebenfalls auf Instanzebene durchgeführt werden [3]. Wenn eine laufende Instanz auf ein geändertes Schema migriert werden soll, muss der aktuelle Zustand der Instanz berücksichtigt werden. Die Verträglichkeit wird über die *Ausführungshistorie* der Instanz bestimmt. Die Ausführungshistorie gibt an welche Aktivitäten gestartet und welche beendet wurden. Jede Aktivität schreibt einen Start-Eintrag, sobald sie gestartet wurde. Genauer gesagt wenn die Aktivität vom Zustand *activated* in den Zustand *running* übergeht. Wenn eine Aktivität erfolgreich beendet wurde, d.h. von *running* nach *completed* wechselt, wird ein entsprechender Endeintrag geschrieben. Eine Instanz gilt als verträglich mit einem neuen Schema, wenn ihre Ausführungshistorie auch mit dem neuen Schema erzeugt werden kann [3].

Im Falle eines Schleifendurchlaufes ist diese Bedingung allerdings zu restriktiv. Wenn eine Instanz bereits einen Schleifendurchlauf nach altem Schema durchlaufen hat, wäre sie nicht mehr verträglich mit einem Schema, bei welchem eine Modifikation innerhalb dieser Schleife stattgefunden hat. Die Ausführungshistorie wird deshalb reduziert, so dass nur der aktuelle Schleifendurchlauf, oder im Falle, dass die Schleife bereits verlassen wurde, nur der letzte Durchlauf betrachtet wird.

Die Überprüfung auf Verträglichkeit sollte möglichst effizient sein. Dadam und Reichert gelingt dies, indem sie die Semantik der Änderungsoperationen berücksichtigen. Für das Einfügen einer neuen Aktivität werden andere Kontrollen durchgeführt, als für das Ändern einer Kontrollflusskante. Für jede elementare Änderungsoperation wird eine Liste von Regeln definiert, die überprüft werden müssen [3]. Wenn alle Regeln erfüllt sind, dann ist diese Änderungsoperation mit der Instanz verträglich. Für diese Kontrollen wird vor allem der aktuelle Zustand der Aktivitäten berücksichtigt, sowie die Einträge in der Ausführungshistorie. Wenn z. B. eine Aktivität eingefügt werden soll, darf keine der nachfolgenden Aktivitäten bereits laufen, ansonsten wird diese Instanz als nicht verträglich mit der Änderung eingestuft. Änderungen in der Reihenfolge von Aktivitäten werden ebenfalls auf diese Weise überprüft.

Für Datenflussänderungen muss die Ausführungshistorie erweitert werden. Wenn eine Aktivität lesend oder schreibend auf ein Datenelement zugreift, muss dies zusätzlich noch vermerkt werden. Die Verträglichkeitsprüfung im Falle einer Datenflussänderung läuft nach dem gleichen Prinzip. Für jede elementare Operation, wird überprüft, ob die entsprechenden Regeln erfüllt sind.

Beispielsweise darf eine Aktivität nicht lesend auf ein Datenelement zugreifen, wenn sie bereits läuft [3].

Nach einer Änderung müssen die Zustände der Aktivitäten der laufenden Instanz neu bewertet werden. Wenn eine Aktivität bereit zur Ausführung ist und eine neue Aktivität vor dieser Aktivität eingefügt wurde, so ist diese nicht mehr bereit zur Ausführung. Der Zustand dieser Aktivität muss nun zurück auf *not activated* gesetzt werden. Eventuell kann es nötig werden, bestimmte Workitems wieder aus den Worklists der Nutzer zu entfernen.

Um den Aufwand zu reduzieren, reicht es aus, nur bestimmte Knoten und Kanten zu betrachten. Welche dies sind, hängt ebenfalls von der verwendeten Änderungsoperation ab. Dadam und Reichert definieren für jede Änderungsoperation die Menge der neu zu betrachtenden Knoten und Kanten. Beispielsweise müssen beim Einfügen einer neuen Aktivität alle Nachfolgerknoten, sowie alle einlaufenden Kanten zu der neuen Aktivität, berücksichtigt werden. Sie präsentieren ebenfalls einen Algorithmus, mit dessen Hilfe dann die eigentliche Neubewertung erfolgt [3].

Gelegentlich kann es erforderlich werden, dass Änderungen rückgängig gemacht werden müssen. Dafür sind Änderungshistorien erforderlich. Es werden zwei unterschiedliche Änderungshistorien für jede laufende Instanz gleichzeitig verwaltet. Einmal der Graph $P_{all,}$ der die aktuelle Struktur und den aktuellen Zustand der WF-Instanz darstellt. Dieser Graph beinhaltet sowohl die temporären als auch die permanenten Änderungen. Außerdem wird noch der Graph P_{Perm} angelegt. Dieser stellt den Graphen dar, der aus dem ursprünglichen Schema nach Anwendung aller permanenten Änderungen hervorgegangen ist. Temporäre Änderungen, sowie der aktuelle Zustand der Instanz, werden hier nicht berücksichtigt. Bei temporären Änderungen genügt es, die Verträglichkeit mit dem Graphen P_{all} zu überprüfen. Wenn permanente Änderungen durchgeführt werden sollen, müssen zusätzliche Aspekte berücksichtigt werden. Beispielsweise darf keine Datenabhängigkeit zwischen einer permanent und einer temporär eingefügten Aktivität bestehen. Aus diesem Grund müssen alle permanenten Änderungen noch zusätzliche gegen den Graphen P_{Perm} getestet werden [7, S.16].

Mit diesem Verfahren läßt sich effizient überprüfen, ob eine laufende Instanz mit einem geänderten Schema verträglich ist. Es bleibt allerdings noch zu klären, wie mit nicht verträglichen Instanzen verfahren werden soll. Dadam und Reichert schlagen eine verzögerte Migration vor [3], d.h. es wird gewartet bis die Instanz einen Zustand erreicht hat, bei dem eine Migration möglich wäre. Es wird dabei unterschieden zwischen Instanzen, die niemals verträglich werden können *(never-more-compliant)* und solchen, die sich gerade in einem Schleifendurchlauf[2] befinden und vielleicht beim nächsten Durchlauf migriert werden können *(re-compliant).* Für erstere Gruppe werden verschiedene Vorgehensweisen vorgeschlagen. Die einfachste Methode wäre es, diese Instanzen unverändert weiterlaufen zu lassen. Alternativ wäre es auch möglich sie teilweise

[2] Dies ist der einzige mögliche Fall, der genannt wird.

zurückzusetzen, bis ein verträglicher Zustand erreicht worden ist. Im schlimmsten Falle würde dies bedeuten, den ganzen bisherigen Verlauf komplett zurückzusetzen. ADEPT bietet ferner die Möglichkeit an *success dependencies* zu definieren. Wenn beispielsweise eine Aktivität gelöscht wird, werden alle davon abhängigen nachfolgenden Aktivitäten ebenfalls gelöscht. Dieser Ansatz ist dem *spheres of control* Konzept sehr ähnlich [7, S. 15], welches auch in WS-BPEL zum Einsatz kommt. Für diejenigen Instanzen, welche als *re-compliant* eingestuft wurden, wird ein Eintrag in einer ECA-Tabelle (Event, Condition, Action) angelegt. Hier ist vermerkt nach welchem Ereignis, welche Bedingung erfüllt sein muss, damit eine Migration möglich wird [2,3].

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

Abbildung 2 : Prinzip der verzögerten Migration.
Quelle : Rinderle, Dadam, Reichert. *Supporting Workflow Schema Evolution by efficient compliant checks*.[2]

In der obigen Abbildung ist ein Auszug aus einer ECA-Tabelle dargestellt. Wenn die aktuelle Schleife erneut ausgeführt wird (Event = Loop_Back), wird die Instanz I_2 auf das neue Schema S' migriert.

3.3 Bewertung

Meiner Meinung nach haben Dadam und Reichert schon sehr ausgereifte Strategien entwickelt. Für die Verträglichkeitsprüfungen wurden bereits effiziente Algorithmen entwickelt [2]. Das Konzept, dass für jede Änderungsoperation eine Menge von Regeln erfüllt sein muss, halte ich für sehr vielversprechend und praktikabel. Alerdings habe ich den Eindruck, dass die Herangehensweise sehr auf Sicherheit ausgelegt ist. Ich vermute deshalb, dass einige dieser Regeln in der Praxis zu restriktiv sein könnten. Das heißt, dass eventuell Instanzen, die migriert werden könnten, als nicht verträglich eingestuft würden. Dies müsste allerdings noch näher untersucht werden.

Die Neubewertung der Zustände, nach einer Änderungsoperation, wurde meiner Ansicht nach sehr effizient gelöst. Dies wird dadurch erreicht, dass anstelle des gesamten Graphen nur eine ausgewählte Menge von Knoten und Kanten betrachtet wird. Hierfür wird ebenfalls schon ein Algorithmus präsentiert. Die Hauptproblematik besteht aber generell in der Behandlung von nicht verträglichen Instanzen. Es wird kein Algorithmus präsentiert, mit dem bestimmt werden kann, ob eine Instanz *never-more-compliant* oder *re-compliant* ist. Außerdem wird nicht aufgezeigt, welche verschiedenen Bedingungen für die ECA-Tabelle möglich sind. Als einziges Beispiel werden lediglich Schleifendurchläufe genannt [2, 3].

Ich vermute, dass der Schleifendurchlauf auch das einzige Kriterium ist, nachdem entschieden wird, ob eine Instanz *never-more-compliant* oder *re-compliant* ist.
Es wird hier also lediglich ein Verfahren präsentiert mit dem nicht verträgliche Instanzen, welche sich gerade in einem Schleifendurchlauf befinden, behandelt werden können. Für andere nicht verträgliche Instanzen werden keine neuen Strategien vorgestellt. Außerdem darf nicht vernachlässigt werden, dass bei dem Ansatz der verzögerten Migration bei jedem Navigationschritt überprüft werden muss, ob die Bedingungen in der ECA-Tabelle erfüllt sind. Bei einer großen Anzahl von Instanzen kann dies zu Leistungseinbußen führen.

Außerdem muss noch berücksichtigt werden, dass bei dem hier vorgestellten Ansatz auf bestimmte strukturelle Prüfungen gänzlich verzichtet wird. Beim Einfügen einer neuen Kontrollflusskante wird beispielsweise nicht überprüft, ob dadurch ein Zyklus erzeugt wird. Dadam und Reichert verfolgen generell den Ansatz, dass die strukturelle Problematik, d.h. Deadlocks oder Lost Updates, komplett vom Nutzer verborgen werden soll. Dem Nutzer sollen Änderungsoperationen auf hohem semantischen Niveau angeboten werden. Diese werden dann intern auf eine Sequenz von Basisoperationen übertragen [24, S.4; 1, S.11]. Auf diese Weise soll vermieden werden, dass der Nutzer Deadlocks oder andere strukturelle Inkonsistenzen überhaupt erst erzeugen kann. Für semantisch hohe Änderungsoperationen werden allerdings nur wenige Beispiele präsentiert. Ein Beispiel ist eine Operation vom Typ „insert X between A and B“, welche die Aktivität X zwischen den Aktivitäten A und B einfügt [7, S.11]. Meiner Ansicht nach muss noch überprüft werden, ob eine Änderungssprache auf hohem semantischen Niveau die Flexibilität bietet, die in der geschäftlichen Praxis benötigt wird. Außerdem bleibt abzuwarten, ob sie auf die Bedürfnisse des Endbenutzers zugeschnitten werden kann, d.h. einerseits mächtig genug ist alle Änderungen durchzuführen, andererseits darf der Nutzer nicht mit zu komplexen Konstrukten konfrontiert werden. Aus meiner Sicht ist dies ein sehr wichtiger Aspekt und es ist noch viel Forschung auf diesem Gebiet nötig.

4. Alternative Ansätze um Instanzen auf geänderte Schemata zu migrieren

Im folgenden sollen alternative Ideen für die Realisierung von Laufzeitmodifikationen an Workflows präsentiert werden. Es wird vor allem auf die Migration von laufenden Instanzen auf geänderte Schemata eingegangen.

4.1 Lösungsvorschläge von Sadiq

Sadiq verwendet einen petrinetzbasierten Ansatz. Eine Modifikation an einem Workflow wird als Austauschen eines Teilgraphen betrachtet. Die Verträglichkeit ist hier über ein *Trace* definiert, d.h. wenn die *Ausführungssequenz* der laufenden Instanz auch mit dem neuen Schema erzeugt werden kann, dann ist sie verträglich. Die Idee, die hierbei zugrunde liegt, ist ähnlich zu dem Ansatz über die Ausführungshistorie, der von Dadam und Reichert vorgeschlagen wird. Allerdings stellt die Ausführungssequenz bei Petrinetzen eine Sequenz von Zuständen dar, die bis zum aktuellen Zeitpunkt bereits durchlaufen wurden. Für eine formale Definition wird auf die Quelle verwiesen [16]. Wohingegen die Ausführungshistorie ein Protokoll ist, in welchem vermerkt wird, dass eine Aktivität gestartet bzw. beendet wurde. Eine strikte Graphenäquivalenz wird von der Autorin abgelehnt, da dies in den meisten Fällen zu restriktiv sei.

Für die dynamische Modifikation von Workflow-Schemas und anschließende Migration der laufenden Instanzen schlägt sie ein 3-phasiges Vorgehen vor. In der ersten Phase werden die Änderungen definiert (Defining the Modification) [16, S.9]. Sie werden durch ein Sequenz von Änderungsoperationen beschrieben, welche in einem Transaktionskontext ausgeführt werden. Auf diese Weise soll gewährleistet werden, dass ein Teilgraph in einem einzigen Schritt ausgetauscht wird. Dies ist für ihren Ansatz erforderlich. Daraufhin wird mittels Graphenanalyse überprüft, ob der ausgetauschte Teilgraph die strukturelle Korrektheit des Workflows erhält. Hierfür wurde der Prototyp FlowMake

implementiert. Bei FlowMake handelt es sich um ein graphisches Tool, welches strukturelle Inkonsistenzen, wie z. B. Deadlocks, auffinden kann.

In der zweiten Phase werden die laufenden Instanzen, gemäß ihrer Verträglichkeit mit dem neuen Schema, gruppiert (Conforming to the modification) [16, S.10]. Die Gruppierung erfolgt in drei Stufen und weitgehend automatisch. In der ersten Stufe werden alle Instanzen identifiziert, die mit dem modifizierten Schema verträglich sind. Diese können unmittelbar migriert werden. In der zweiten Stufe werden die übrigen unverträglichen Instanzen näher betrachtet. Sie werden nach ihrem aktuellen Stadium (*stage*) gruppiert. Ziel ist es hierbei zu bestimmen, ob die Instanzen automatisch oder manuell bearbeitet werden. Es ist möglich, dass Instanzen bereits so weit fortgeschritten sind, dass eine Anpassung zu aufwändig wäre oder einem Abort gleichkäme. Diese müssen dann durch einen Menschen bewertet und das Vorgehen individuell bestimmt werden. In der dritten Stufe werden die Instanzen in Klassen eingeteilt. Die Instanzen einer Klasse werden dann auf einen *Compliance Graphen* migriert, welcher mit der entsprechenden Klassen assoziiert wurde. Hierbei handelt es sich um einen hybriden Graphen, der eine Brücke zwischen dem alten und dem neuen Schema darstellt. Die Instanz durchläuft einen Pfad, welcher teilweise aus dem alten Schema, dem Compliance Graphen und dem neuen Schema besteht. Der Compliance Graph algorithmisch erzeugt werden. Für nähere Details wird auf die Quelle verwiesen [16]. Der Compliance Graph beinhaltet Kompensationsaktivitäten und wird an einer geeigneten Stelle in den Workflow eingefügt. Die Idee hierbei ist, dass einige Aktivitäten kompensiert werden und der Workflow so in einen verträglichen Zustand zurückversetzt wird.

In der dritten Phase wird die eigentliche Migration vorgenommen (Effectuating the modification) [16, S.11]. Es muss festgelegt werden, wie mit bereits gestarteten Aktivitäten verfahren werden soll. Es ist möglich sie zu Ende laufen zu lassen (Flush) oder abzubrechen (Abort). Die laufenden Instanzen müssen ferner mit Sperren versehen werden und nach der Migration wieder fortgesetzt werden.

4.2 Lösungsvorschläge von Ellis

Ellis verwendet ebenfalls einen petrinetzbasierten Ansatz und Änderungen werden ebenfalls als Austauschen eines Teilgraphen betrachtet. Das Vorgehen ist dem Ansatz von Sadiq sehr ähnlich. Allerdings unterscheidet es sich in manchen Punkten von dem Ansatz von Sadiq.

Zuerst wird die *Change Region* bestimmt, das ist der Teilgraph, welcher alle Aktivitäten beinhalten, die von der Änderung betroffen sind. Die Change Region vor der Änderung wird als *Old Region* bezeichnet. Die Change Region auf die alle Änderung angewandt wurden, wird als *New Region* bezeichnet. Durch die Änderung wird also die Old Region durch die New Region ersetzt.

Tokens, die sich außerhalb der Change Region befinden, sind von der Änderung nicht betroffen. Um die Instanzen zu bearbeiten, deren Tokens sich gerade in der Change Region befinden, muss noch eine weitere Eigenschaft bestimmt werden.

Die Change Region kann entweder ein *Upsizing* oder ein *Downsizing* vornehmen. Im Falle eines *Upsizings* stellt die New Region ein Erweiterung der Old Region dar. Dies bedeutet, dass jede Ausführungssequenz der Old Region auch eine mögliche Ausführungssequenz der New Region sein muss. Das bedeutet ferner, dass jeder gültige Zustand der Old Region auf einen gültigen Zustand der New Region abgebildet werden kann. Das Gegenstück dazu ist das *Downsizing*, hier kann nicht jede Ausführungssequenz der Old Region mit der New Region repliziert werden. Für eine formale Definition wird auf die Quellen verwiesen [13].

Es werden nun zwei Vorgehensweisen für die Migration vorgeschlagen einmal ein *immediate change*, hierbei wird die Old Region unmittelbar durch die New Region ersetzt. Die Tokens im Petrinetz werden entsprechend transferiert. Dieser Ansatz bietet sich an wenn ein Upsizing durchgeführt wird. Die Korrektheit dieses Ansatzes wird bewiesen. Es ist auch intuitiv nachvollziehbar, dass im Falle eines Upsizing eine unmittelbare Migration möglich ist, da der neue Graph das Verhalten des alten exakt reproduzieren können muss. Das andere Verfahren wird als *Synthetic Cut-Over Change* bezeichnet. Die Migration erfolgt nun dadurch, dass hybride Graphen gebildet werden, in welchen die Old Region mit der New Region verknüpft wird. Dieses Verfahren ist prinzipiell mit *Upsizing* und *Downsizing* möglich. Im Falle eines *Upsizing* kann die neue Region über *flow jumpers* direkt mit der alten überlagert werden (siehe Abbildung 3).

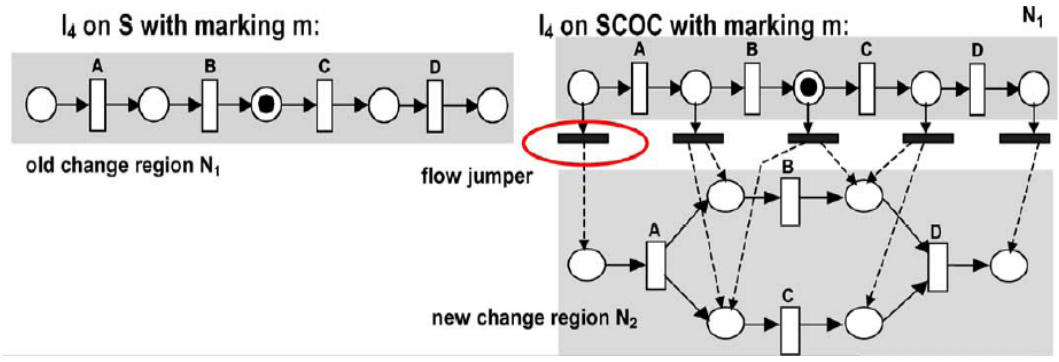

Abbildung 3 : Upsizing mit Synthetic Cut-Over Change
Quelle : Rinderle, Reichert, Dadam. *Correctness criteria for dynamic changes in workflow systems – a survey*. [10]

In der obigen Abbildung ist der Fall dargestellt, dass die Aktivitäten B und C durch eine Änderung nun auch parallel ausgeführt werden können. Über *flow jumpers* können nun die Tokens der Old Region N_1 in die New Region N_2

transferiert werden. Jeder flow jumper bildet einen Zustand der Old Region in einen äquivalenten Zustand der New Region ab.

Im Falle eines Downsizing wird der neue Teilgraph parallel zu dem alten eingefügt. Beide Teilgraphen teilen sich ein gemeinsames Input- und Outputfeld und sind gleichzeitig in dem Gesamtgraphen präsent. Die Tokens, die sich zum Zeitpunkt der Änderung bereits in der Change Region befunden haben, laufen nach der Old Region weiter. Die Tokens, welche die Change Region erst betreten, werden bereits gemäß der New Region bearbeitet [13, 10 S.21].

4.3 Lösungsvorschläge von Casati

Casati benutzt zur Beschreibung von Workflows einen Ansatz, der dem von Petrinetzen ähnlich ist. Der Workflow wird durch einen Graphen dargestellt. In Abbildung 4 sind die Elemente, aus denen ein Workflow-Graph aufgebaut werden kann, dargestellt **:** task, total fork, conditional fork, iterative join, start/stop symbol (von links nach rechts).

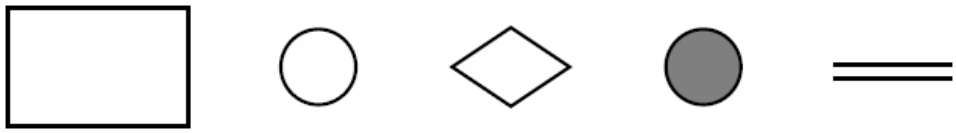

Abbildung 4 : Graphische Repräsentation von Workflow-Elementen.
Quelle : Fabio Casati and Stefano Ceri and Barbara Pernici and Giuseppe Pozzi : *Workflow Evolution.* [12]

Es ist ferner möglich Variablen und Conditions zu definieren, welche aber nicht graphisch dargestellt werden. Jede Aktivität kann Werte in Variablen schreiben und von dort auch wieder auslesen. Bei einem totalen Fork werden alle auslaufenden Kanten weiterverfolgt, während bei einem conditional fork, abhängig von einer Condition nur bestimmte Pfade weiterverfolgt werden. Bei einem totalen Join müssen die Conditions aller einlaufenden Kanten erfüllt sein. Während dies bei einem iterativen Join nicht unbedingt der Fall sein muss. In der Abbildung 5 ist ein Beispielworkflow dargestellt, der eine PC-Manufaktur abbildet.

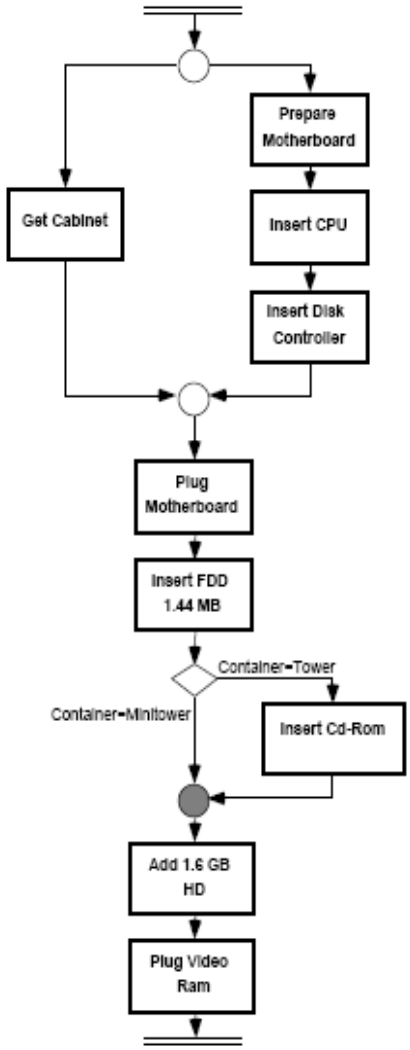

Abbildung 5 : Beispielworkflow.
Quelle : Fabio Casati and Stefano Ceri and Barbara Pernici and Giuseppe Pozzi : *Workflow Evolution.* [12]

Casati bezeichnet das Problem der Migration als *Dynamic workflow evolution.* Dabei beschränkt er sich auf die strukturellen Aspekte von Ad hoc-Änderungen. Bei Datenbanken werden für Schemamodifikationen nur bestimmte Operationen zugelassen, welche Invarianten beibehalten. Diese Idee versucht Casati auf Workflows zu übertragen. Er definiert hierfür eine *Workflow Manipulation Language* (WFML), die aus einer Menge von Änderungsprimitiven besteht, zusammen mit deren syntaktischer und semantischer Beschreibung [12]. Die Menge dieser Änderungsoperationen ist vollständig, d.h. jeder gültige Workflow kann in jeden anderen gültigen Workflow überführt werden. Ferner sind die Änderungsoperationen konsistent, d.h. jede Operation überführt einen korrekten Workflow in einen anderen korrekten Workflow. Da jede einzelne Änderungoperation die Korrektheit des Workflows erhält, erhält auch jede Aneinanderreihung davon die Korrektheit. Ferner ist die Menge der Operationen

auch minimal, d.h. keine ihrer Teilmengen erfüllt die ersten beiden Bedingungen. Die Änderungsprimitive lassen sich in zwei Klassen einteilen. Eine davon ist die Klasse der *declaration primitives*. Mit diesen Operationen können Variablen und Aktivitäten hinzugefügt bzw. entfernt werden. Die *flow primitives* stellen die zweite Klasse dar. Mit ihnen kann die Struktur eines Workflows modifiziert werden. Auf diese Art können Kontrollflusskanten umgesetzt werden, sowie deren Conditions modifiziert werden.

Die Migration wird dadurch realisiert, dass die Änderungen, welche am Schema vorgenommen wurden, auch an den laufenden Instanzen durchgeführt werden. Der Verträglichkeitsbegriff ist identisch mit dem Ansatz von Dadam und Reichert. Eine Instanz gilt als verträglich mit einem Schema, sofern die bisherige Ausführungshistorie auch mit dem neuen Schema erzeugt werden kann. Die Verträglichkeit mit einem geänderten Schema hängt also im Wesentlichen davon ab, wie weit die laufende Instanz bereits fortgeschritten ist. Casati unterscheidet zwei Arten von Migrationen : *Conditional* und *Unconditional*. Wenn die laufende Instanz mit dem geänderten Schema verträglich ist, kann eine *Unconditional* Migration durchgeführt werden, d.h. die Instanz kann unmittelbar und automatisch auf das neue Schema migriert werden, weitere Maßnahmen zur Konsistenzsicherung sind nicht erforderlich. Schwieriger verhält es sich mit nicht verträglichen Instanzen. Hier wird die *Conditional* Migration angewandt. Dies bedeutet, dass die laufenden Instanzen zuerst in einen verträglichen Zustand gebracht werden müssen. Dafür müssen Aktivitäten rückgängig gemacht werden, so lange bis ein kompatibler Zustand erreicht wurde. Allerdings kann es unter Umständen nicht möglich sein, einzelne Aktivitäten wieder rückgängig zu machen. In diesem Fall müssen die betroffen Instanzen von einem Workflow Administrator (WFA) analysiert werden und jeweils geeignete Verfahren ausgewählt werden. Am einfachsten zu realisieren sind das Abort- oder Flush-Verfahren. Ferner schlägt Casati vor, die nicht verträglichen Instanzen auf einen hybriden Graphen zu migrieren (migration to ad-hoc workflow) [12. S.13]. Dieser beinhaltet bereits möglichst viele der Änderungen, ist allerdings zu einer möglichst großen Zahl laufender Instanzen kompatibel, welche mit dem neuen Schema nicht verträglich sind. Auf diese Weise sollen laufende Instanzen bereits von Änderungen profitieren können, obwohl sie mit dem geänderten Schema nicht verträglich sind. Der hybride Graph muss im Vorfeld von dem WFA manuell erstellt worden sein. Das System kann den WFA unterstützen, indem die laufenden Instanzen automatisch gruppiert werden. Die Gruppen können daraufhin den jeweiligen hybriden Graphen zugeordnet werden, zu denen sie kompatibel sind.

4.4 Zusammenfassung und Bewertung

Die obigen Ansätze sind in ihrer Grundstruktur sehr ähnlich. Alle Autoren fordern eine Änderungssprache, der eine Menge von Basisänderungsoperationen zu grunde liegt. Die Menge der Operationen soll minimal, konsistent und vollständig sein. Casati gibt für diese Operationen bereits eine Beschreibung der Syntax und

der Semantik an [12]. Dem Nutzer sollen Änderungsoperationen auf hohem semantischen Niveau angeboten werden. Diese werden dann intern auf eine Sequenz dieser Basisoperationen abgebildet. Bei Sadiq wird dem Nutzer die Möglichkeit geboten, mittels eines graphischen Tools (FlowMake) einen Teilgraphen auszutauschen.

Für die Definition der Verträglichkeit liegt bei allen Autoren die gleiche Idee zu grunde. Die Verträglichkeit wird bei allen Autoren über die Ausführungshistorie, bzw. Ausführungssequenz bei petrinetzbasierten Ansätzen, automatisch bestimmt.

Eine weitere Gemeinsamkeit ist es, die laufenden Instanzen bezüglich ihrer Verträglichkeit zu gruppieren. Instanzen, die verträglich sind, sollen automatisch migriert werden. Die nicht verträglichen Instanzen werden weiter unterteilt. Für jede dieser Gruppen wird nun unterschiedlich verfahren. Es werden hierfür verschiedene Vorgehensweisen vorgeschlagen, z. B. Abort oder Versionierung. Welche Vorgehensweise jeweils am besten geeignet ist muss bei allen Ansätzen von einem Menschen ermittelt werden.

Eine weitere Gemeinsamkeit bei allen Ansätzen ist es, unverträgliche Instanzen, für welche ein Abort oder Flush nicht möglich ist, auf einen hybriden Graphen (Sadiq : Compliance Graph) zu migrieren, so dass laufenden Instanzen bereits von Prozessoptimierungen profitieren können, obwohl sie mit dem neuen Schema nicht verträglich sind. Ellis skizziert einen Ansatz, wie der alte Graph mit dem neuen überlagert werden kann (Synthetic Cut-Over Change). Allerdings muss hierfür erst der betroffene Bereich (Change Region) erst bestimmt werden. Ellis weist explizit daraufhin, dass dies nicht trivial ist.

Unterschiede bestehen darin, welches Modell zur Darstellung von Workflows verwendet wird. Sadiq und Ellis verwenden Petrinetze während Casati ein eigenes graphisches Modell präsentiert.

Ein weiterer Unterschied ist der Einsatz von WFA. Bei Casati steht die nicht-automatische Bearbeitung stark im Vordergrund, das System nimmt lediglich eine unterstützende Funktion ein. Die hybriden Graphen müssen alle im Vorfeld von einem WFA erstellt worden sein. Wohingegen Sadiq einen Ansatz skizziert, mit dem der Compliance Graph automatisch erstellt werden kann. Der Ansatz von Ellis zielt ebenfalls auf eine weitgehende Automatisierung ab. Allerdings ist das große Problem die Lokalisierung der Change Region. Hierfür ist kein effizienter Algorithmus verfügbar. Weshalb eine automatische Bearbeitung der nicht verträglichen Fälle nur sehr schwer möglich ist.

Aus meiner Sicht haben die Autoren sehr viel versprechende Ansätze entwickelt, welche alle mit fundierten Formalismen untermauert werden. Die Verträglichkeit von laufenden Instanzen mit geänderten Schemata kann bei allen Ansätzen automatisch bestimmt werden. Allerdings wird die Frage, wie mit nicht verträglichen Instanzen verfahren werden soll, nur unzureichend beantwortet. Da der Ansatz von Casati stark auf nicht-automatische Bearbeitung abzielt, ist er sicherlich am einfachsten zu realisieren. Allerdings ist für eine wirtschaftliche

Anwendung ein weitgehend automatisierter Ansatz am interessantesten. Weshalb dieser Ansatz noch um weitere Konzepte zu Automatisierung erweitert werden muss. Ellis und Sadiq präsentieren bereits Ansätze in dieser Richtung, die allerdings nicht ganz unproblematisch sind. Das Hauptproblem bei dem Ansatz von Ellis ist das Identifizieren der Change Region. Hierfür ist kein effizienter Algorithmus verfügbar. Eine weitere Schwierigkeit ist die Generierung der flow jumpers. Es ist meiner Meinung nach ein semantisches Problem zu bestimmen, welche Zustände in der alten Region mit Zuständen in der neuen Region äquivalent sind. Aus meiner Sicht kann dies nur in sehr einfachen Fällen automatisch erfolgen. In der Regel wird dafür menschliches Eingreifen benötigt. Der Ansatz von Sadiq sieht zwar vor, dass unverträgliche Instanzen auf einen hybriden Compliance Graph migriert werden, welcher automatisch erstellt werden kann. Allerdings besteht dieser lediglich aus Kompensationsaktivitäten, welche die Instanz in einen verträglichen Zustand zurückversetzten. Auf diese Weise kann unter Umständen sehr viel Arbeit verloren gehen.

Ein weiterer Nachteil von petrinetzbasierten Verfahren (Ellis, Sadiq) ist die mangelnde Trennung von Daten- und Kontrollfluss. Viele Änderungen in der geschäftlichen Praxis betreffen aber lediglich den Datenfluss. Dies kann mit diesen Verfahren nur unzureichend gelöst werden. Deshalb sind diese Verfahren aus meiner Sicht für einen anwendungsnahen Ansatz noch nicht geeignet.

5. Migration von geänderten Instanzen auf geänderte Schemata

Es ist prinzipiell möglich Änderungen auf Instanz- oder auf Schemaebene durchzuführen. Wenn nun eine Instanz durch eine Ad hoc-Änderung modifiziert wurde und zu einem späteren Zeitpunkt das zu grunde liegende Schema modifiziert wurde, dann entsteht eine Reihe weiterer Probleme. Ein wichtiger Aspekt sind strukturelle Probleme. Beispielsweise können bei der Migration Deadlocks entstehen. Dies ist in Abbildung 6 dargestellt. Auf Schemaebene wurde zwischen D und E ein Übergang eingefügt. Bei der laufenden Instanz I wurde zwischen F und C ein Übergang eingefügt. Beide Änderungen sind, für sich alleine genommen, korrekt und unproblematisch. Wenn aber I auf das geänderte Schema migriert wird, entsteht ein Deadlock.

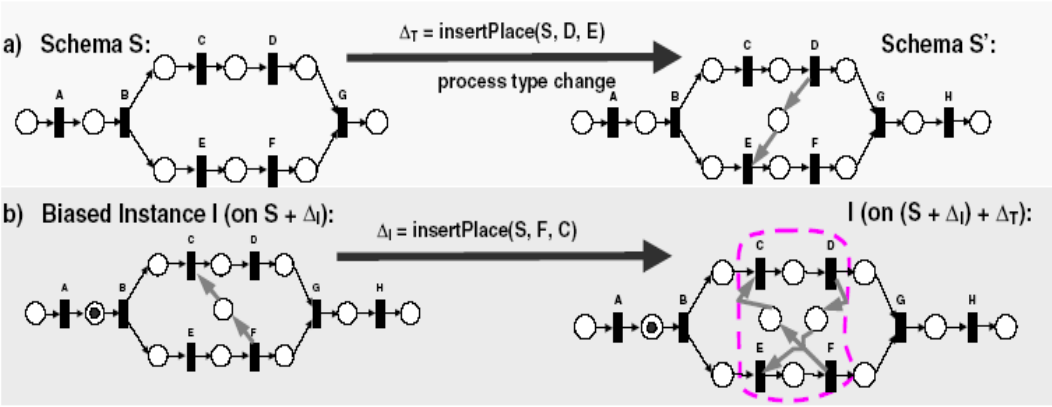

Abbildung 6 : Struktureller Konflikt in Petri-Netzen (Deadlock)
Quelle : Rinderle, Reichert, Dadam. *On Dealing with Structural Conflicts between Process Type and Instance Changes***.** [8]

Rinderle, Dadam und Reichert präsentieren nun einen Ansatz wie diese Problematik erkannt werden kann. Hierfür werden die Änderungen auf Instanz- und Schemaebene in zwei separaten Listen gespeichert, diese werden als

Änderungshistorien bezeichnet. Sie beinhalten jeweils die Sequenz von Basisänderungsoperationen, welche die gewünschte Modifikation realisiert. Rinderle, Dadam und Reichert stellen nun einen Algorithmus vor, der diese Änderungshistorien miteinander vergleicht und überprüft, ob bei einer Migration der Instanz auf das Schema ein Deadlock entstehen würde [8, S.281]. Eine weiteres Problem ist es, dass neue Aktivitäten eingefügt werden können. Wenn nun diese neuen Aktivitäten bei einem Deadlock beteiligt wären, würde der Algorithmus dies nicht erkennen, da die Aktivitäten nur bei einem der Graphen vorhanden wären.

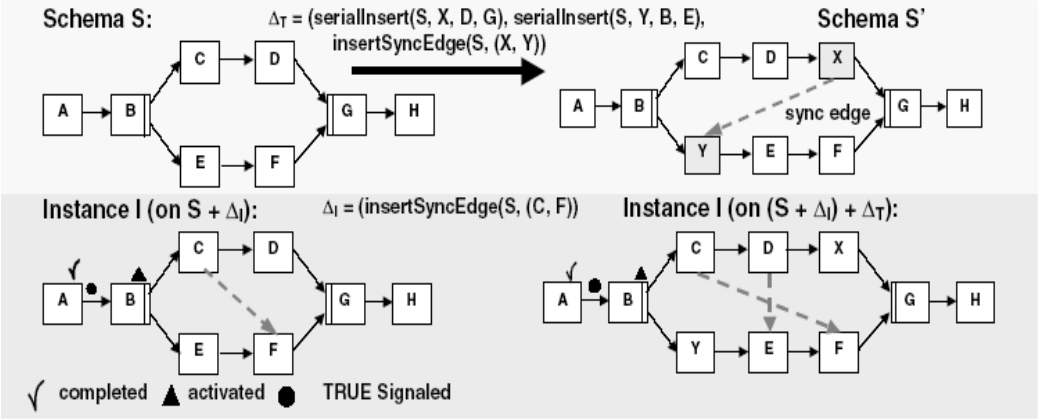

Abbildung 7 : Einfügen von Synchronisationskanten
Quelle : Rinderle, Reichert, Dadam. *On Dealing with Structural Conflicts between Process Type and Instance Changes***.** [8]

Um diese Problematik zu beherrschen, wird ein Algorithmus vorgeschlagen, der virtuelle Synchronisationskanten erzeugt, die ausschließlich zwischen Aktivitäten verlaufen, die sowohl in dem Schema als auch in der Instanz vorhanden sind [8, S. 283]. Dabei wird die ausgehende Synchronisationskante einer neuen Aktivität auf ihre Vorgängeraktivität umgesetzt. Die einlaufende Synchronisationskante einer neuen Aktivität wird auf ihren Nachfolger umgelegt. Dies ist in Abbildung 7 skizziert. Hier verläuft auf der Schemaebene eine Synchronisationskante zwischen X und Y. Diese Aktivitäten sind in der Instanz nicht präsent. Durch den Algorithmus wird diese Synchronisationskante zu der virtuellen Synchronisationskante zwischen D und E umgeformt. Auf diese Weise kann dann der Algorithmus korrekt überprüfen, ob ein Deadlock vorliegt oder nicht. Diese virtuellen Synchronisationskanten dienen lediglich zur Überprüfung und sind im tatsächlichen Workflow nicht enthalten.

Um Datenflussfehler, z. B. fehlende Input-Daten, zu bestimmen wird ebenfalls ein Algorithmus präsentiert. Die Idee hierbei ist die gleiche, wie beim vorherigen Problem. Die Änderungen, bezüglich des Datenflusses, an dem Schema und der Instanz werden verglichen, um potentielle Fehler zu erkennen [8, S.284]. Die Autoren weisen allerdings daraufhin, dass dieser Ansatz in manchen Fällen zu restriktiv ist. Es kann vorkommen, dass Änderungen als problematisch erkannt würden, obwohl keine Datenflussfehler bei einer Migration auftreten würden. Als

Alternative wird hier vorgeschlagen, die Änderungen tatsächlich durchzuführen und umfangreiche Datenflussanalysen vorzunehmen. Dies ist allerdings nicht sehr effizient und sollte nur in Ausnahmefällen durchgeführt werden.

Ein weiteres Problem ist es, dass durch Löschen von Kontrollflusskanten Aktivitäten isoliert werden können. Dies ist problematisch, da keine isolierten Aktivitäten vorhanden sein dürfen, um klar definierte Start- und Endzustände des Workflows zu garantieren. Es sind zwar ebenfalls Tests möglich, die isolierte Aktivitäten erkennen, aber diese Tests werden überflüssig, indem dem Nutzer lediglich Änderungsoperationen auf semantisch hohem Niveau angeboten werden. Dadurch soll es unmöglich werden einzelne Aktivitäten zu isolieren [8].

Ein weiterer wichtiger Aspekt ist, dass sich Änderungen überlappen können. Beispielsweise kann auf Schemaebene eine neue Aktivität eingefügt werden. Die gleiche Änderung kann aber bereits durch eine Ad hoc-Änderung auf Instanzebene durchgeführt worden sein. Wenn nun diese Instanz auf das neue Schema migriert würde, würde die neue Aktivität zweimal[3] eingefügt werden. Dies entspricht nicht unbedingt der Absicht des Nutzers. Außerdem kann es vorkommen, dass durch strukturelle Änderungen auf der Instanzebene, Änderungen der Schemaebene nicht mehr anwendbar sind[4]. Wenn beispielsweise auf Schemaebene eine Aktivität an eine andere Stelle gesetzt werden soll, kann der Fall vorkommen, dass diese Aktivität auf der Instanzebene bereits gelöscht wurde. Eine Migration dieser Instanz auf das neue Schema wäre also nicht möglich [9, S.102]. Es muss also der Grad der Überlappung zwischen den Änderungen auf Schema- und der Instanzebene bestimmt werden.

Ein erster Ansatz diese Problematik in den Griff zu bekommen ist es, lediglich kommutative Änderungen zuzulassen. Für eine formale Definition wird auf die Quellen verwiesen [9]. Die Idee ist, dass kommutative Änderungen unabhängig von der Reihenfolge der Schema- und der Instanzänderungen sind und daher in äquivalenten Graphen resultieren. Äquivalenz bedeutet hier, dass die gleichen Ausführungshistorien erzeugbar sind. Außerdem dürfen die Änderungen nicht die gleichen Aktivitäten einfügen. Wenn dies erfüllt ist, dann sind sie disjunkt[5] und eine Migration ist möglich. Allerdings stellt dies eine sehr große Einschränkung dar. Es muss also ein Verfahren entwickelt werden, um den Grad der Überlappung möglichst genau zu bestimmen.

Rinderle, Reichert und Dadam schlagen zwei unterschiedliche Vorgehensweisen vor. Einen strukturellen und einen operationalen Ansatz. Die Vorgehensweisen sind noch sehr theoretisch und die Grundideen werden lediglich skizziert [9].

---

[3] Dies hängt von der Änderungssprache ab. In diesem Fall gehen die Autoren von einer Operation der Form „Füge A zwischen B und C ein“ aus, welche die Aktivität A an einer beliebigen Stelle zwischen den Aktivitäten B und C einfügt. Eine Operation, welche die Aktivität A direkt zwischen B und C einfügt, könnte nicht zweimal hintereinander ausgeführt werden.

[4] Bei ADEPT sind Migrationen dadurch realisiert, dass die Änderungen des Schemas auf die Instanz angewandt werden

[5] Damit Änderungen disjunkt sind, müssen beide Bedingungen erfüllt sein : Sie müssen kommutativ sein und dürfen nicht die gleichen Aktivitäten einfügen.

Für einen strukturellen Ansatz werden die resultierenden[6] Graphen miteinander verglichen, dies wird als *Delta Analysis* bezeichnet. Als Grundlage dient hier das Vererbungskonzept von Van der Aalst. Dieses Konzept wird im nächsten Kapitel näher erläutert. Allerdings hat dieser Ansatz Probleme bei Änderungen, die die Reihenfolge der Aktivitäten ändern. Hier kann der Algorithmus lediglich überlappende Änderungen erkennen, allerdings können keine Aussagen über den Grad der Überlappung gemacht werden. Dies ist im Allgemeinen nicht präzise genug. Der operationale Ansatz hat diese Schwäche nicht.

Der operationale Ansatz besteht darin, die Änderungshistorien der Änderungen auf Instanz- und Schemaebene direkt miteinander zu vergleichen. Allerdings müssen die Änderungshistorien dafür erst bereinigt werden. Zuerst müssen kompensierende Änderungen entfernt werden. Wenn beispielsweise eine Aktivität eingefügt wird und danach wieder gelöscht wird. Solche Änderungen heben sich gegenseitig auf und können deshalb aus der Änderungshistorie entfernt werden. Als nächstes müssen versteckte Änderungen kenntlich gemacht werden. Wenn eine Aktivität gelöscht wird und an anderer Stelle wieder eingefügt wird, hat dies den gleichen Effekt, als ob die Aktivität an die neue Stelle verschoben worden wäre. In der Änderungshistorie wird dies entsprechend vermerkt. Ferner müssen noch überschriebene Änderungen entfernt werden. Wenn eine Aktivität verschoben wird und daraufhin erneut verschoben wird, so ist der Effekt der ersten Verschiebung komplett neutralisiert. Sie kann dann aus der Änderungshistorie entfernt werden. Danach erfolgt der Vergleich der beiden Änderungshistorien. Die große Schwäche eines operationalen Ansatzes liegt allerdings darin Änderungen zu erkennen, die aufeinander aufbauen [9].

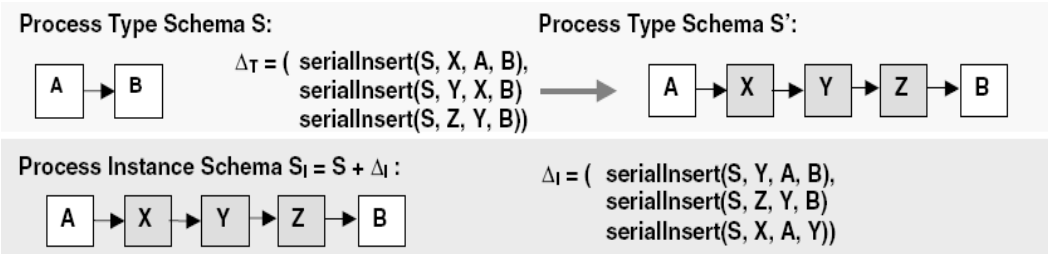

Abbildung 8 : Äquivalente Schema- und Instanzänderungen.
Quelle : Rinderle, Reichert, Dadam. *Disjoint and Overlapping Process Changes: Challenges, Solutions, Applications.* [9]

In Abbildung 8 wird dies verdeutlicht. Die Änderungshistorien auf Instanz- und Schemaebene sind komplett unterschiedlich. Sie resultieren aber in dem gleichen Graphen. Mit dem operationalen Ansatz würde lediglich erkannt werden, dass diese Änderungen nicht disjunkt sind. Allerdings würde nicht erkannt werden, dass diese Änderungen äquivalent sind.

[6] Es werden drei Graphen miteinander verglichen. Das unveränderte, ursprüngliche Schema, das ursprüngliche Schema nachdem die Schemaänderungen darauf angewandt wurden, sowie das ursprüngliche Schema nachdem die Instanzänderungen darauf angewandt wurden.

Jeder Ansatz hat seine Stärken und Schwächen. Es wird deshalb ein hybrider Ansatz vorgeschlagen. Hier wird mittels eines Algorithmus eine bereinigte Änderungshistorie erzeugt (operationaler Ansatz). Ferner werden die Mengen der eingefügten, gelöschten und bewegten Aktivitäten bestimmt (struktureller Ansatz). Es wird hierfür auch ein Algorithmus angegeben [9, S.117]. Diese Informationen werden dann dazu benutzt den Grad der Überlappung zu bestimmen. Allerdings wird nicht näher darauf eingegangen, wie dies tatsächlich erfolgen soll. Außerdem wird kein Lösungsvorschlag gemacht, wie mit den laufenden Instanzen verfahren werden soll, wenn der Grad der Überlappung erst bestimmt ist. Überdies wurde kein Formalismus angegeben wie der Grad der Überlappung beschrieben werden kann.

Aus meiner Sicht sind den Autoren bereits beachtliche Fortschritte gelungen. Sie präsentieren als einzige Ansätze, wie veränderte Instanzen auf veränderte Schemata migriert werden können. Allerdings ist dieser Bereich sehr komplex und es sind noch viele Probleme zu lösen. Vor allem wird nicht erwähnt wie weiter verfahren werden soll, wenn der Grad der Überlappung tatsächlich bestimmt wurde. Außerdem wird auf semantische Probleme nicht eingegangen. Beispielsweise kann die Überlappung ja vom Benutzer gewollt sein, z. B. kann es erforderlich sein für einen besonders anspruchsvollen Kunden mehrere Qualitätskontrollen im Workflow einzufügen, auch wenn bereits eine Kontrolle durch eine Schemaänderung eingefügt wurde. Meiner Meinung nach, sind noch sehr viele Probleme zu lösen, bevor es möglich sein wird ein WFMS zu implementieren, welches sowohl Änderungen auf Instanz- und Schemaebene gleichermaßen unterstützt.

6. Objektorientierter Ansatz, um Verträglichkeit zu garantieren

Van der Aalst schlägt einen anderen Ansatz vor, um mit nicht verträglichen Instanzen umzugehen. Für das Problem, dass laufende Instanzen nicht mit einem geänderten Schema verträglich sind, verwendet er die Begriffe *Dynamic Change Problem* oder *Dynamic Change Bug*.

Sein Ansatz ist es, nur bestimmte Änderungen zuzulassen. Dadurch wird das Dynamic Change Problem zwar nicht gelöst, aber umgangen. Er verfolgt dafür einen objektorientierten Ansatz. Die Grundideen der Objektorientierung, wie Vererbung, Kapselung und Abstraktion, werden auf Workflows übertragen. Kapselung bedeutet nun, dass Aktivitäten geblockt werden können. Abstraktion wird dadurch erzielt, dass bestimmte Aktivitäten ignoriert werden können. Er definiert überdies spezielle Vererbungsrelationen, die mit dedizierten *Inheritance-preserving Transformation Rules* erhalten werden. Die Objektklassen entsprechen den Workflowschemata, wohingegen die abgeleiteten Objekte die einzelnen Instanzen darstellen. Er kann beweisen, dass wenn eine seiner Vererbungsrelationen zwischen Schema und Instanz gilt, es keinen Dynamic Change Bug gibt [18, 23].

Für die Darstellung der Workflows benutzt Van der Aalst Petrinetze [18, 19, 20, 23]. Als Neuerung wird die *Silent Action* eingeführt. Das ist eine Aktivität, die keine sichtbaren Resultate erzeugt. Aktivitäten werden in die Silent Action umbenannt, um Abstraktion zu realisieren. Die dynamischen Aspekte eines Workflows werden durch Attribute beschrieben. Im Falle von Petrinetzen sind dies die Tokens. Es wird ferner vorgeschlagen gefärbte Tokens zu verwenden. Auf diese Weise können den Tokens zusätzliche Informationen, z. B. Kundendaten, zugewiesen werden [19]. Wenn Petrinetze für die Modellierung von Workflows verwendet werden, müssen sie strukturell korrekt sein, d.h. keine Deadlocks aufweisen und korrekt terminieren können. Diese Eigenschaften können in Polynomialzeit überprüft werden.

Sein Ansatz basiert im Wesentlichen auf dem Prinzip der Vererbung. Vereinfacht ausgedrückt ist ein Workflow X eine Unterklasse eines andern Workflows Y, wenn jede Ausführungssequenz von Y auch auf X erzeugbar wäre. Das heißt X kann alles was Y auch kann, aber eventuell auch mehr. Es werden vier unterschiedliche Arten von Vererbungsrelationen definiert. Um zu überprüfen, ob eine Vererbungsrelation zwischen zwei gegebenen Workflows gilt, werden einzelne Aktivitäten entweder geblockt (Kapselung) oder in die Silent Action (Abstraktion) unbenannt. Daraufhin wird überprüft, ob beide Graphen die gleichen Ausführungssequenzen durchlaufen. Die möglichen Vererbungs-relationen sind :

Protocol Inheritance : Wenn nur die Aktionen, die in der Superklasse vorhanden sind bei der Unterklasse betrachtet werden und diese dann das gleiche leistet. Um dies zu überprüfen, können einzelne Aktivitäten geblockt werden. Wenn daraufhin die Ausführungssequenz der Superklasse auch von der Unterklasse reproduziert werden kann, dann gilt diese Eigenschaft.

Projection Inheritance : Wenn nur die Auswirkungen derjenigen Aktivitäten betrachtet werden, die in beiden Graphen vorhanden sind und diese dann die gleichen Ausführunssequenzen erzeugen, dann gilt diese Relation. Um dies zu überprüfen werden die neu eingefügten Aktivitäten in die Silent Action umbenannt.

Protocol/Projection Inheritance : Diese Relation gilt, wenn sowohl Protocol Inheritance als auch Projection Inheritance gilt.

Life-cycle Inheritance : Das Verhalten der Superklasse kann von dem abgeleiteten Objekt reproduziert werden, indem die neu eingefügten Aktivitäten entweder geblockt oder in die Silent Action umbenannt werden.

Diese Relationen sind alle reflexiv, antisymmetrisch und transitiv.

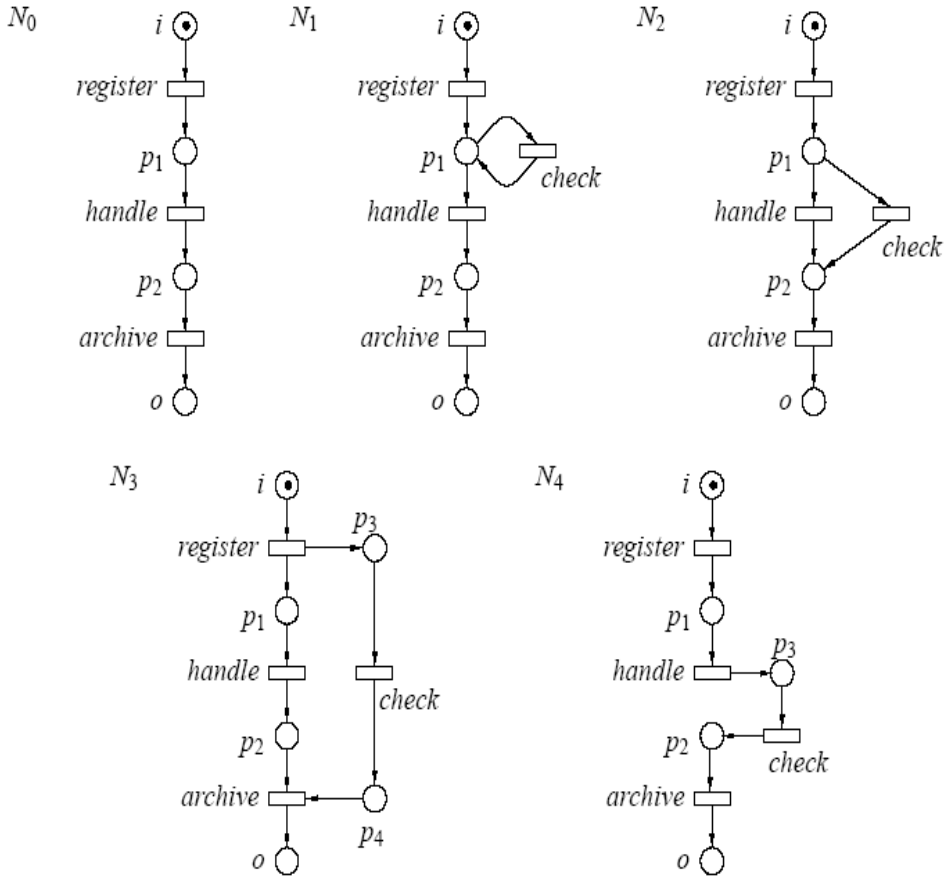

Abbildung 9 : Fünf Workflow-Definitionen.
Quelle : Van der Aalst, Basten. *Inheritance of Workflows An approach to tackling problems related to change*. [18]

Die Vererbungsrelationen werden mit Abbildung 9 verdeutlicht. Die Graphen N_1 bis N_4 stellen Ableitungen des Graphen N_0 dar. Bei N_1 kann die neue Aktivität *check* beliebig häufig iteriert werden. Bei N_2 kann sie anstelle der Aktivität *handle* durchgeführt werden. In N_3 wird *check* parallel zu *handle* durchgeführt und in N_4 muss *check* direkt nach *handle* ausgeführt werden.

Die *Protocol Inheritance* gilt bei den Graphen N_1 und N_2. Wenn *check* geblockt wird, ist die Ausführungshistorie äquivalent zu N_0. Für N_4 gilt dies allerdings nicht. Wenn *check* geblockt wird, dann resultiert dies in einem Deadlock.

Bei dem Graphen N_3 ist die *Projection Inheritance* erfüllt. Wenn *check* ignoriert wird, d.h. in die *Silent Action* umbenannt wird, ist das Verhalten dieses Graphen nicht von N_0 zu unterschieden.

Die *Life-cycle Inheritance* ist bei allen Graphen erfüllt.

Es werden vier verschiedene Transformationsregeln (Inheritance-preserving Transformation Rules*)* definiert. Diese sollen Modifikationen an den Graphen ermöglichen und gleichzeitig die Vererbungsrelationen bewahren. Auf diese

Weise bleibt jeder Zustand der Superklasse auf einen äquivalenten Zustand der Unterklasse stets abbildbar[7]. Die Grundideen dieser Änderungsoperationen werden hier lediglich skizziert, für Details wird auf die Quellen verwiesen [18]. Die Grundidee hinter allen Änderungsoperationen ist, dass ein korrekter Graph an bestimmten Stellen des aktuellen Graphen eingefügt werden kann.

PPS : Diese Operation erhält die Protocol/Projection Inheritance. Das Verhalten, dass von einem neuen Subgraphen definiert wird, kann mehrmals iteriert werden, bevor zu der ursprünglichen Ausführung des Ursprungsgraphen zurückgekehrt wird. Es können also Schleifen hinzugefügt werden.

PTS : Diese Modifikation erhält die Protocol Inheritance. Sie ermöglicht es alternative Zweige einzuführen, deren Einstiegs- und Ausstiegspunkte im Ursprungsgraphen liegen.

PJS : Diese Änderung erhält die Projection Inheritance. Es wird das sequentielle Einfügen von Aktivitäten ermöglicht. Die Idee ist, dass eine Kante durch einen Graphen ersetzt werden kann.

PJ3S : Hierdurch wird die Life-cycle Inheritance erhalten. Es wird beschrieben wie parallele Zweige einfügt werden.

Für jede dieser Regeln sind entsprechende Umkehrtransformationen ebenfalls definiert.

Entsprechen werden auch Regeln eingeführt, um die Zustände (*Markings*) der Petrinetze auf die abgeleiteten Unterklassen zu transferieren. Im Falle einer PPS, PTS oder PJS Transformation ist dies einfach. Die Tokens können direkt transferiert werden. Da durch diese Transformationen die alten Zustände erhalten bleiben, es kommen ausschließlich neue Zustände hinzu.

Im Falle einer PJ3S Transformation ist dies allerdings nicht mehr so einfach. Da bestehende Zustände im neuen Graphen nun eventuell kein aquivalentes Pendant mehr haben. Hier wird ein spezielles Vorgehen benötigt. Es wird hier zuerst ein *virtueller Place* eingeführt, der dem neuen parallelen Zweig entspricht. Die Tokens in diesem Place können nun auf zwei verschiedene Arten transferiert werden. Eine Möglichkeit ist die *Conservative* Methode. Die Tokens werden auf den Input-Place des parallelen Zweiges gesetzt. Das Gegenstück dazu stellt die *Progressive* Methode dar. Hier werden die Tokens auf den Output-Place des parallelen Zweiges gesetzt. Diese beiden Verfahren unterscheiden sich in semantischer Hinsicht. Bei der Conservative Methode wird das migrierte Petrinetz in einen Zustand versetzt, als ob der neue parallele Zweig erst noch durchlaufen werden muss. Bei der Progressive Methode wird das Petrinetz in den Zustand versetzt, als ob der neue Zweig bereits durchlaufen wurde. [18].

[7] Übertragen auf Workflows bedeutet dies, dass eine Migration immer möglich ist, wenn eine der Vererbungsrelationen gilt.

Durch dieses Konzept wird das Dynamic Change Problem nicht gelöst, aber vermieden. Wenn die laufenden Instanzen eines Schemas Unterklassen gemäß den Vererbungsrelationen sind, kann stets migriert werden. Dies wird formal bewiesen [18]. Das bedeutet, dass eine Migration einer laufenden Instanz auf ein geändertes Schema immer möglich ist, sofern man sich auf die vier vorgestellten Änderungsoperationen beschränkt.

Aus meiner Sicht ist dieser Ansatz vielversprechend. Hervorzuheben ist, dass er mit einem fundierten Formalismus untermauert wird. Allerdings sind noch einige Probleme zu lösen. Die erste Frage ist, wie effizient festgestellt werden kann, ob zwischen zwei beliebigen Graphen eine Vererbungsrelation gilt. Van der Aalst schlägt hier lediglich einen aufwändigen Ansatz vor. Die neuen Aktivitäten zu blockieren bzw. auszublenden und dann zu testen, ob die Ausführungssequenzen des alten Graphen erzeugbar sind.

Ein weiteres Manko ist meiner Meinung nach, dass sich durch die Änderungsoperationen lediglich korrekte Graphen einfügen lassen, d.h. der Graph muss stets terminieren können und darf keine strukturellen Fehler aufweisen. Das hat zur Folge, dass dieser erst separat generiert und getestet werden muss, bevor er eingefügt werden kann. Meiner Meinung nach besteht die Gefahr, dass die Nutzer dadurch überfordert werden und deshalb ein entsprechendes System ablehnen werden.

Ein weiteres generelles Problem von petrinetzbasierten Verfahren ist, wie bereits erwähnt, die mangelnde Trennung von Daten- und Kontrollfluss. Dies tritt hier natürlich ebenfalls auf.

Überdies werden mit diesem Ansatz keine Änderungen gestattet, mit denen sich die Reihenfolge der Aktivitäten ändern läßt [9]. Diese Problematik wurde bereits im vorherigen Kapitel angesprochen. Vor allem in einem klinischen Umfeld ist es aber unbedingt erforderlich, dass die vorgegebene Reihenfolge von Aktivitäten geändert werden kann, um einen Notfall zu bearbeiten [6, 24]. Dies stellt eine sehr große Schwäche dieses Verfahrens dar. Meiner Meinung nach ist dieser Ansatz deshalb noch nicht für praktische Anwendungen geeignet.

7. Anforderungen in verteilten Umgebungen

In der geschäftlichen Praxis kommen monolithische Hard- und Softwaresysteme nur noch sehr vereinzelt vor. Sie wurden zunehmend durch verteilte leistungsfähigere Systeme ersetzt. Für Laufzeitmodifikationen an Workflows ergibt sich dadurch eine Vielzahl neuer Probleme. ADEPT*Distribution* bietet hierfür eine Reihe von Lösungen an [5].

Ein WFMS kann aus mehreren WF-Servern bestehen. Eine Instanz kann abschnittsweise von unterschiedlichen Servern kontrolliert werden. Allerdings gibt es in verteilten Systemen keine globalen Informationen, anhand derer entschieden werden könnte, ob Änderungen, aufgrund des aktuellen Zustands der Instanz, zulässig sind oder nicht. Es wäre zwar möglich globale Informationen anzulegen und diese von einem einzelnen Server verwalten zu lassen. Allerdings würde dieser Server dann einen single point of failure darstellen, der die Verfügbarkeit des gesamten Systemes einschränken würde. In ADEPT*Distribution* wurden einige Ansätze implementiert, die ohne eine zentrale Kontrolle auskommen. Dennoch sind alle Änderungen möglich, die auch in einem nicht verteilten System möglich wären. Im folgenden sollen diese Ideen und Ansätze kurz skizziert werden.

Bei ADEPT*Distribution* wird ein Workflow partitioniert. Die Partitionen werden dann unterschiedlichen Servern zugewiesen (siehe Abbildung 10). Sofern ein Server einen Abschnitt vollständig bearbeitet hat, wird der aktuelle Zustand der Ausführung an den nachfolgenden Server übermittelt[8]. Dies erfolgt dadurch, dass die Ausführungshistorie übermittelt wird.

[8] Bauer, Reichert und Dadam verwenden hierfür ebenfalls den Begriff „Migration“. Dies ist nicht zu verwechseln mit der Migration von laufenden Instanzen auf ein geändertes Schema. Im Rahmen dieser Studienarbeit wird der Begriff „Migration“ ausschließlich für Anpassung von Instanzen an geänderte Schemata verwendet.

Parallele Zweige laufen unabhängig voneinander, d.h. ein Server besitzt keine Informationen über Server, die Zweige bearbeiten, die parallel zu seinem eigenen laufen. Auf diese Weise wird der Aufwand zur Übertragung der Ausführungshistorien reduziert.

Sofern menschliche Bearbeiter bei der Ausführung involviert sind, sollten die Server in den Teilnetzen liegen, in denen sich die meisten Bearbeiter befinden. Auf diesem Weise kann die Kommunikation effizienter gestaltet werden, da das lokale Netz nicht verlassen werden muss. Dies ist relativ problemlos, sofern die Bearbeiter bereits statisch im Vorfeld bestimmt wurden. Komplizierter wird es bei abhängigen Bearbeiterzuweisungen, z. B. „Aktion X soll von dem gleichen Bearbeiter wie Aktion Y durchgeführt werden". ADEPT*Distribution* bietet hierfür die Möglichkeit der variablen Serverzuweisungen, z. B. („Server im Teilnetz des Bearbeiters von Aktion C").

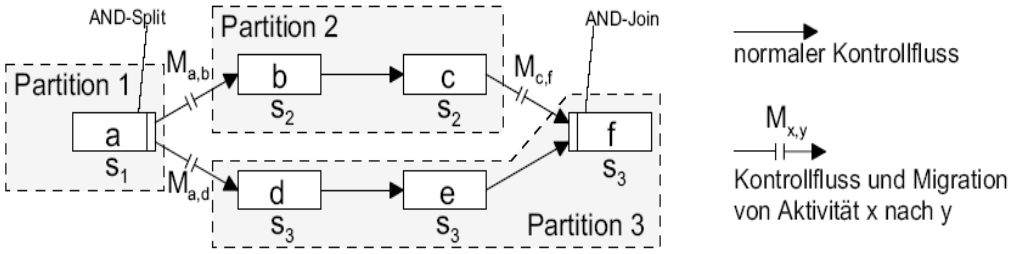

Abbildung 10 : Partitionierung eines WF – Graphen und verteilte Ausführung
Quelle : Bauer, Reichert, Dadam. *Adaptives und verteiltes Workflow-Management.* [5]

Reichert und Dadam präsentieren einen Ansatz, mit dem das Änderungskonzept von Adept auf verteilte Systeme übertragen werden kann. Komplexe Änderungsoperationen werden ebenfalls auf eine sequentielle Folge von Basisoperationen zurückgeführt. Mittels Regeln werden Vorbedingungen überprüft, ob diese Operationen auf die Instanz anwendbar sind. Diese Änderungsoperationen werden daraufhin in der Änderungshistorie eingetragen. Zusätzlich dazu wird noch in der Ausführungshistorie ein Vermerk eingetragen, dass eine Änderung stattgefunden hat. Letzteres findet in nicht verteilten Systemen normalerweise nicht statt. Sofern eine Änderung durchgeführt werden soll, müssen Zustandsinformationen von anderen Servern eingeholt werden. Ein Broadcast an alle Server wäre zu aufwändig, da viele Server im System unter Umständen gar nicht an der modifizierten Instanz beteiligt sind. Vorteilhafter ist es, nur diejenigen Server anzusprechen, die aktuell an der Instanz arbeiten. Die Server, die bereits ihre Arbeit an der laufenden Instanz abgeschlossen haben, werden nicht mehr benötigt, da die benötigten Informationen bereits in der Ausführungshistorie und der Änderunghistorie vermerkt wurden und diese bereits an nachfolgende Server weitergegeben wurde.

Es ist unvermeidlich, das ein Server über alle dynamischen Änderungen informiert sein muss, die bisher an der Instanz durchgeführt wurden. Deshalb wäre es vorteilhaft, zusätzlich zu den aktuell beteiligten Servern, auch diejenigen

zu ermitteln, die in der Zukunft noch an der aktuellen Instanz beteiligt sein werden. Dies ist aber bei variablen Serverzuweisungen unter Umständen gar nicht möglich, da im Moment noch gar nicht bestimmt werden kann, welcher Server in Zukunft beteiligt sein wird. Aus diesem Grund müssen Änderungen auch in der Ausführungshistorie vermerkt werden, damit nachfolgende Server Informationen über stattgefundene Änderungen erhalten[9]. Es können also lediglich die Server ermittelt werden, die aktuell an der Instanz beteiligt sind. Dies ist allerdings ebenfalls nicht trivial, da ein Server keine Informationen über Server besitzt, die gerade an parallelen Zweigen arbeiten.

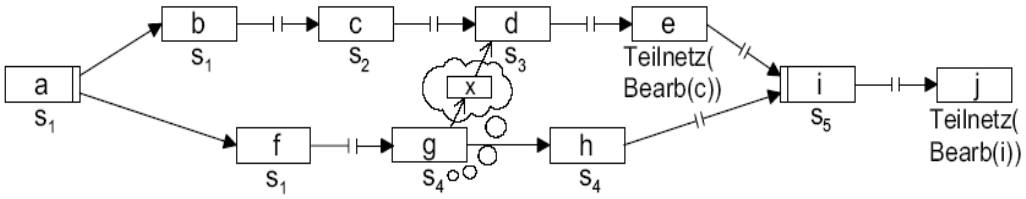

Abbildung 11 : Einfügen einer neuen Aktivität durch einen Server
Quelle : Bauer, Reichert, Dadam. *Adaptives und verteiltes Workflow-Management.* [5]

Im obigen Beispiel möchte der Server S_4 die Aktivität x zwischen den Aktivitäten g und d einfügen. Er besitzt allerdings keine Kenntnis wie weit die Verarbeitung des oberen Zweiges bereits fortgeschritten ist. Insbesondere besitzt er keine Informationen darüber, welcher Server gerade aktiv ist.

Es muss also eine Art „Verzeichnis“ geben, in welchem jeweils die aktuellen Server eingetragen sind. Ein zentraler Server kommt dafür nicht in Frage, da dieser Server einen Flaschenhals darstellen würde. Reichert und Dadam schlagen nun einen *Server Manager* vor, der die Menge der aktiven Server verwalten soll. Der *Server Manager* ist jeweils derjenige Server, der eine Instanz startet. Unterschiedliche Instanzen können also unterschiedlichen Server Manager zugewiesen sein. Es wird auch ein Algorithmus präsentiert, mit dem die Menge der aktiven Server aktuell gehalten werden kann [5, S.10]. Hierbei fordern die Server, wenn sie einen Abschnitt beenden, Sperren bei dem Server Manager an und übermitteln ihm an welchen Server die Arbeit abgegeben wird. Wenn nun ein Server eine dynamische Änderung an der Prozessinstanz durchführen möchte, muss er beim Server Manager eine Sperre anfordern. Dadurch wird verhindert, dass unterschiedliche Server gleichzeitig Änderungen an der Instanz vornehmen und dadurch eventuell eine fehlerhafte Instanz entsteht. Dieser Fall könnte eintreten, wenn in Abbildung 11 beispielsweise der Server S_3 eine Aktivität zwischen d und h einfügen möchte, während der Server S_4 die Aktivität X zwischen g und d einfügt. Aus Sicht jedes einzelnen Servers wären die Änderungen unproblematisch, da die Änderung des anderen Servers noch nicht präsent ist. Beide Änderungen zusammen erzeugen einen Zyklus.

[9] Wenn ein Server eine Partition übernimmt, erhält er die gesamte Ausführungshistorie der Instanz vom vorherigen Server.

Reichert und Dadam machen noch einen Vorschlag, wie die Übermittlung der Änderungshistorien effizienter gestaltet werden kann. Ein Server kann unter Umständen mehrere Male bei der Bearbeitung einer Instanz beteiligt sein. Er verfügt über die Änderungshistorie bis zu dem letzten Abschnitt, den er bearbeitet hat. Es genügt also, wenn der Server bei seinem Vorgänger lediglich die Einträge anfordert, die seit seiner letzten Beteiligung gemacht wurden. Dadam und Reichert haben einen Algorithmus erarbeitet, der genau dies leistet [5, S.15].

Aus meiner Sicht ist es Dadam und Reichert gelungen die Probleme, die in einem verteilten System bestehen, zu lösen. Es ist besonders hervorzuheben, dass ihr Ansatz ohne einen zentralen Server, und damit ohne einen single point of failure, auskommt. Es werden schon sehr ausgereifte und effiziente Algorithmen präsentiert, die aus meiner Sicht ohne größere Probleme implementiert werden können. Allerdings darf nicht übersehen werden, dass hier lediglich eine Abbildung des Verhaltens eines verteilten Systems auf ein nicht verteiltes System dargestellt wird. Die Probleme, welche die Laufzeitmodifikationen von Instanzen in einem nicht verteilten System betreffen, z. B. unverträgliche Instanzen, werden durch diese Ansätze weder gelöst noch vermieden. Deshalb müssen zuerst die Änderungsprobleme generell gelöst werden, bevor man damit beginnen kann derartige Lösungen verteilt zu implementieren.

8. Flexible Workflows

Mangan und Sadiq, von der University of Queensland in Australien, verstehen unter einem flexiblen Workflow einen Geschäftsprozess, der in einem hochgradig dynamischen Umfeld abläuft, und nicht vollständig im Vorfeld definiert werden kann [15]. Als Beispiele werden die Behandlung eines Patienten im Gesundheitswesen oder die Durchführung eines Hochschulstudiums genannt. Die Aktivitäten, die zur Auswahl stehen, sind zwar im Vorfeld bekannt, aber die Reihenfolge ist in der Regel stark unterschiedlich. Flexibilität bedeutet hier das ein partiell oder gar nicht definierter Workflow dennoch ausgeführt werden kann. Mangan und Sadiq präsentieren ein System mit welchem ein Studium an Hochschulen abgebildet werden soll. Da es den Studenten möglich ist, unterschiedliche Kurse in unterschiedlicher Reihenfolge zu belegen, können die Workflows erst während des Studiums, also zur Laufzeit, dynamisch aufgebaut werden. Die einzelnen Lehrveranstaltungen entsprechen den Aktivitäten. Dem Nutzer wird dazu ein Pool angeboten, indem die aktuell möglichen Aktivitäten und Workflow-Konstrukte angeboten werden. Es wurde noch kein Prototyp implementiert, d.h. die Überlegungen sind hier von theoretischer Natur. Die Menge aller Aktivitäten werden durch beschriftete Icons repräsentiert. Wobei dem Nutzer nur diejenigen auf dem Bildschirm angezeigt werden, welche sich gerade in dem Pool befinden. Durch einen Klick auf das entsprechende Icon wird die Aktivität oder das gewünschte Konstrukt dann am Ende des aktuellen Workflows eingefügt. Der Workflow einer laufenden Instanz wird also zur Laufzeit aus vordefinierten Bausteinen zusammengesetzt. Bei der Generierung des Pools wird der aktuelle Status des Workflows ebenfalls berücksichtigt, d.h. durch den aktuellen Zustand ändert sich auch die Zusammensetzung des Pools.

In dem vorgestellten Beispiel beschränken sich die Autoren auf eine kleine Menge von Konstrukten. Es ist möglich Sequenzen von Aktivitäten zu definieren. Außerdem stehen dem Nutzer noch *Fork*- und *Synchronize*- Operationen zur Verfügung. Dies entspricht dem AND- bzw. XOR-Fork, bzw. dem AND-Join oder XOR-Join von ADEPT. Datenflussaspekte werden nicht berücksichtigt. Dadurch, dass dem Nutzer nur diese einfachen Konstrukte zur Verfügung stehen,

soll es vermieden werden, dass bei der Konstruktion strukturelle Probleme, wie z. B. Deadlocks, entstehen können. Es wird außerdem noch vorgeschlagen, dass es dem Nutzer verboten sein sollte XOR-Forks zu verwenden, da ja nur eine einzige Instanz modelliert werden soll und die Entscheidung welcher Pfad ausgeführt werden soll, von dem Modellierer zur Laufzeit erfolgen kann.

Die eigentlichen Konsistenzprüfungen erfolgen durch die Gestaltung des Pools. Die Elemente, die dem Nutzer im Pool zur Verfügung gestellt werden, werden durch drei Arten von Constraints bestimmt :

Selection Constraints : Diese Regeln bestimmen, ob eine Aktivität in den Pool aufgenommen wird und dem Nutzer als Baustein zur Verfügung gestellt wird. Diese Bedingungen lassen sich in drei Klassen einteilen :

1. Vorbedingungen : z. B. kann eine Lehrveranstaltung nur dann ausgewählt werden, wenn die Grundlagenveranstaltung dazu zuvor besucht wurde.
2. Begleitbedingungen : Eine Lehrveranstaltung kann z. B. nur in Verbindung mit anderen ausgewählt werden.
3. Inkompatibilitätsbedingungen : Wenn eine Lehrveranstaltung besucht wurde, kann eine andere unter Umständen nicht mehr besucht werden.

Termination Constraints : Diese Bedingungen überprüfen, inwieweit das Ziel dieses Prozesses erreicht wurde und ob die laufende Instanz erfolgreich abgeschlossen werden kann. Im vorliegenden Beispiel muss eine bestimmte Anzahl an Veranstaltungen besucht sein, bevor das Studium erfolgreich abgeschlossen werden kann. Eine Instanz ist, solange diese Bedingungen noch nicht erfüllt sind, eine offene Instanz. Offene Instanzen sind noch nicht abgeschlossen und können nicht erfolgreich terminieren.

Build Constraints : hierbei handelt es sich um zusätzliche Einschränkungen, die den Aufbau des Workflows betreffen, z. B. kann es sein, dass einige Lehrveranstaltungen im laufenden Semester nicht angeboten werden oder das die Gesamtzahl der Lehrveranstaltungen pro Semester beschränkt ist.

Mangan und Sadiq schlagen ferner das Konzept der *evolving workflows* als Erweiterung ihres Ansatzes vor. Dieser Ansatz sieht vor, dass Workflows nur teilweise definiert werden können. Die Arbeitsabläufe zwischen diesen vordefinierten Teilsegmenten, werden von den beteiligten Personen selbst geregelt, z. B. durch eine Kommuniaktion per eMail. Durch die *Termination Constraints* kann dann stets überprüft werden, ob das Ziel des Geschäftprozesses erreicht wurde.

Magan und Sadiq haben meiner Ansicht nach ein sehr interessantes Konzept entwickelt. Auf diese Weise wird es ermöglicht hochgradig dynamische und unvorhersehbare Geschäftsprozesse mittels eines WFMS abzubilden. Bisher konnten solche Prozesse nur unzureichend mittels Workflows erfasst werden.

Der Ansatz der hier präsentiert wird, zeichnet sich außerdem dadurch aus, dass keine größeren Änderungen am WFMS erforderlich sind. Die eigentlichen Konsistenzprüfungen werden durch die verschiedenen Constraints bei der Generierung des Pools garantiert.

Allerdings ist noch kein Prototyp implementiert, der diese Konzepte implementiert. Vor allem werden keine konkreten Algorithmen präsentiert, mit denen sich der aktuelle Inhalt des Pools bestimmen lässt.

Überdies beschränken sich die Autoren nur auf begrenzte Konstrukte, ob sich damit alle Geschäftsvorfälle effektiv abbilden lassen, bleibt fraglich. Aus meiner Sicht wird es in der betrieblichen Praxis unvermeidlich sein, komplexere Konstrukte, wie Schleifen oder alternative Zweige, zu implementieren. Dadurch entstehen vor allem Probleme, welche die strukturelle Integrität des Workflows betreffen. Es werden dann weitere Prüfungen auf Deadlocks oder Schleifen möglich, was die Komplexität noch wesentlich erhöhen wird.

Des weiteren werden Datenflussaspekte nicht berücksichtigt. Was allerdings einen sehr wichtigen Themenkomplex darstellt und mit weiteren Problemen, wie bereits in vorherigen Kapiteln diskutiert, verbunden ist.

Ferner muss ich dieser Ansatz noch hinsichtlich seiner Benutzerfreundlichkeit bewähren. Es bleibt abzuwarten, ob dieser Ansatz für den Endbenutzer eine wirkliche Erleichterung darstellt. Im vorliegenden Beispiel wird der Benutzer gezwungen jeden Arbeitsschritt einzeln zu planen und manuell in den Workflow einzufügen. Meiner Meinung nach stellt dies keine große Arbeitserleichterung dar, welche WFMS eigentlich bieten sollen.

Eine weitere Schwachstelle dieses Ansatzes ist, dass der Nutzer über viel Prozesswissen verfügen muss. Im Idealfall sollte der Prozess in der strategischen Ebene eines Unternehmens definiert und optimiert werden und der Endbenutzer von Modellierungsaspekten befreit werden. Außerdem ist es nicht möglich dieses Konzept in Umfeldern einzusetzen, bei denen der Workflow geheim gehalten werden sollte. Wir erinnern uns, dass Produkt gleich Prozess ist. Wenn nun jeder Nutzer den kompletten Worklfow zusammensetzen kann, ist eine Geheimhaltung praktisch unmöglich und die Konkurrenz kann leicht Zugriff auf die unternehmensinternen Workflows erhalten.

Insgesamt bin ich der Meinung, dass es sich hierbei um ein sehr interessantes Konzept handelt, dass allerdings noch auf einer sehr frühen Stufe steht. Es sind noch viele Probleme zu lösen, bis letztendlich ein funktionierender Prototyp entwickelt werden kann.

9. Vorabdefinierte Ausnahmen

In der betrieblichen Praxis kommt es häufig zu Ausnahmesituationen, die es erforderlich machen von der vordefinierten Arbeitsabfolge abzuweichen. Nicht selten kommt es vor, dass einige Aktivitäten vorgezogen werden müssen. In einem Krankenhaus ist es z. B. erforderlich, im Notfall, den Patienten sofort zu behandeln und seine Daten erst später zu erfassen. ADEPT bietet die Möglichkeit Ausnahmen bereits während der Modellierung zu berücksichtigen [1, 4]. Auf diese Weise sollen umfangreiche Ad hoc-Änderungen zur Laufzeit vermieden werden.

Es wird die Möglichkeit geboten, Vorwärtssprünge zu planen, d.h. es ist im Vorfeld definierbar, dass eine Aktion unter bestimmten Umständen vorgezogen werden kann. Allerdings ist es erforderlich, dass dies auch als Ausnahme erkennbar ist. Die Autoren schlagen nun vor, das bestehende ADEPT Konzept um Prioritäten zu erweitern. Den Aktivitäten kann entweder die Priorität *regular* oder *exceptional* zugewiesen werden. Standardmäßig werden die Aktivitäten mit *regular* versehen. Sofern Menschen involviert sind, sollten Ausnahmeaktionen auch gesondert dargestellt werden. Denkbar wäre es, Ausnahmen in der Worklist farblich hervorzuheben oder in einem gesonderten Abschnitt aufzuführen (siehe Abbildung 12).

Es ist aber auch erforderlich eine Aktivität, in Abhängigkeit von dem aktuellen Zustand des Workflows, als Ausnahme oder als reguläre Aktivität zu behandeln. Deshalb werden zusätzlich noch Prioritätskanten eingeführt [4]. Diese Kanten sind jeweils mit einer Priorität versehen. Sobald die Quellaktivität erfolgreich terminiert, wird der Zielaktivität die Priorität der Kante zugewiesen. In der Abbildung ist ein Beispiel dargestellt. Zu Beginn ist die Aktivität D im Zustand *exceptional*, d.h. sie kann in Ausnahmefällen vorgezogen werden. Wenn nun im weiteren Verlauf des Workflows die Aktivität C erfolgreich abgeschlossen wird, ändert sich die Priorität von D nach *regular*. Semantisch bedeutet dies, dass die normale Reihenfolge A, B, C, D, E ist. In einer Ausnahmesituation, kann die Reihenfolge auf A, D, B, C, E oder A, B, D, C, E geändert werden.

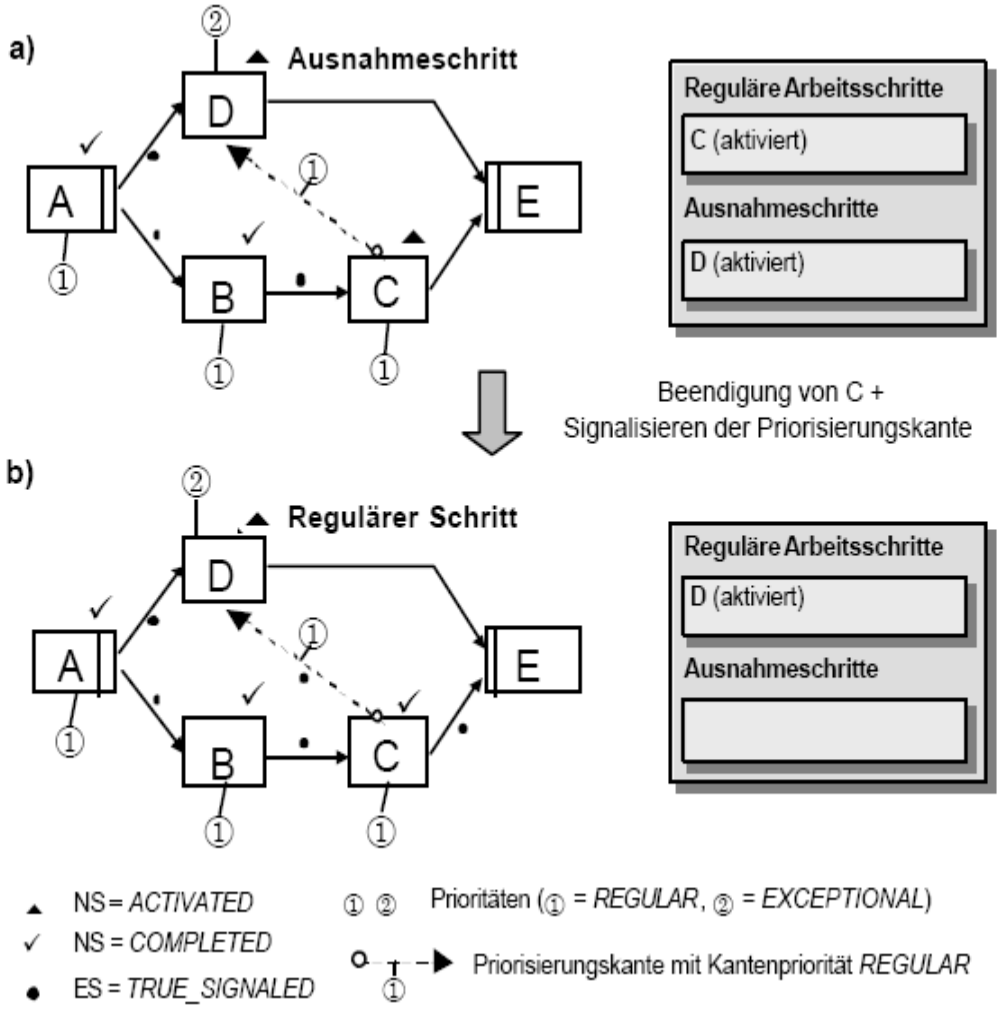

Abbildung 12 : Änderung der Ausführungspriorität eines Aktivitätenknoten
Quelle : Reichert, Bauer, Fries, Dadam. *Modellierung planbarer Abweichungen in Workflow-Management-Systemen.* [4]

Der Modellierer kann diese Vorwärtssprünge durch Shortcut-Kanten modellieren. Diese Darstellung wird dann in eine eindeutige interne Darstellung übersetzt (siehe Abbildung 13).

Es ist zudem noch möglich zu bestimmen, wie mit den übersprungenen Aktivitäten verfahren werden soll. Diese können rückgängig gemacht oder nachträglich ausgeführt werden. Wenn letzteres erfolgen soll, muss eine zusätzliche Aktivität bestimmt werden. Diese dient als Synchronisierungspunkt. Mit der Bearbeitung dieser Aktivität wird erst dann begonnen, wenn die vorgezogenen und übersprungenen Aktivitäten alle ausgeführt wurden. In der Abbildung 13 ist dies dargestellt. Der Modellierer hat eine Shortcut-Kante von A nach F,G eingefügt. Alle Aktivitäten sollen abgeschlossen sein, bevor die Aktivität H begonnen wird.

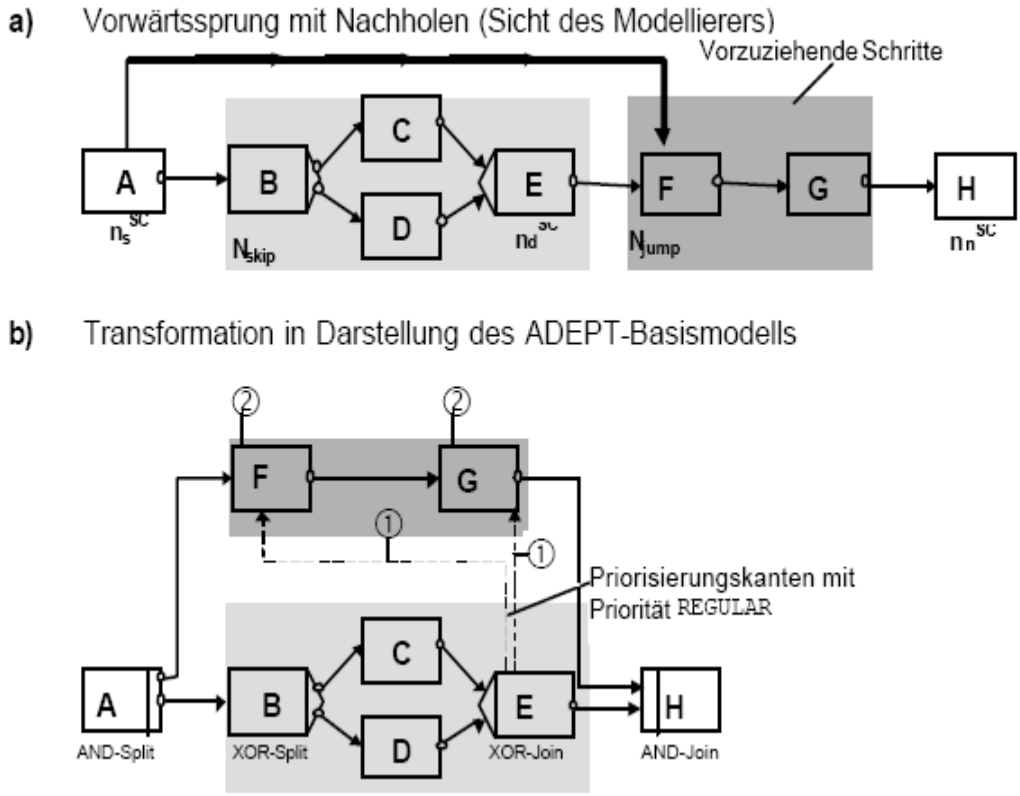

Abbildung 13 : Umsetzen von Vorwärtssprüngen mit Nachholen
Quelle : Reichert, Bauer, Fries, Dadam. *Modellierung planbarer Abweichungen in Workflow-Management-Systemen.* [4]

Die Autoren stellen ebenfalls Ansätze dar, wie Rücksprünge modelliert werden können. Eine spezielle Form von Rücksprüngen stellen Fehlerrücksprünge dar. Dafür werden Fehlerkanten und -codes eingeführt. Jede dieser Kanten wird mit dem Fehlercode ihrer Aktivität verknüpft. Wenn nun dieser Fehlercode während der Ausführung der Aktivität gesetzt wird, wird die Zielaktivität der entsprechenden Fehlerkante ausgeführt. Die Aktivitäten des Rücksetzbereiches werden kompensiert. Dieser Ablauf erfolgt im Allgemeinen automatisch.

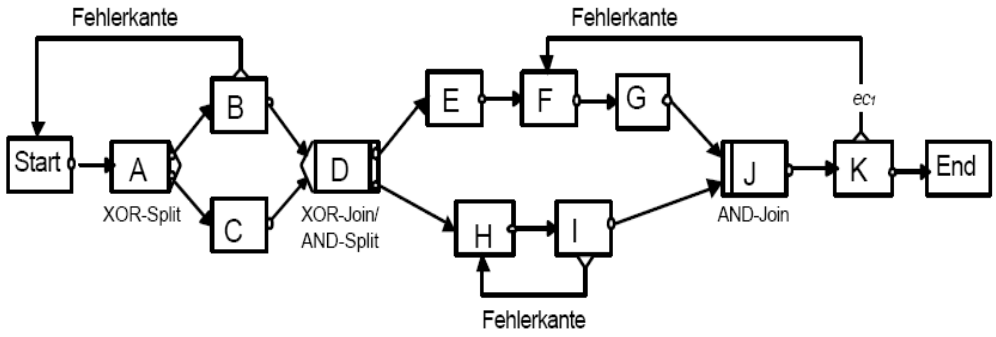

Abbildung 14 : Modellierung von Rücksprüngen durch Fehlerkanten
Quelle : Reichert, Bauer, Fries, Dadam. *Modellierung planbarer Abweichungen in Workflow-Management-Systemen.* [4]

Die Autoren stellen ferner noch *RegainControl*-Kanten vor. Hier kann der Benutzer nach Beendigung einer Aktivität bestimmen, ob der vordefinierte Verlauf weiter verfolgt werden soll oder ob zu dem Ziel der *RegainControl*-Kante zurückgesprungen werden soll. Das Vorgehen ist das gleiche wie bei Fehlerkanten, mit dem Unterschied dass hier der Rücksprung manuell durch einen Benutzer initiiert werden kann.

Reichert, Bauer, Fries und Dadam präsentieren hier einen sehr interessanten Ansatz um Ausnahmefälle bereits bei der Modellierung zu berücksichtigen. Meiner Meinung ist es ein klarer Vorteil, dass der Modellierer relativ einfach durch *Shortcut*-Kanten Ausnahmen modellieren kann. Die Übersichtlichkeit des Schemas wird dadurch kaum beeinträchtig.

Außerdem finde ich es sehr vorteilhaft, dass dieses Konzept keine großen Erweiterungen am bestehenden Workflowmodell benötigt. Es müssen lediglich Priotätskanten und -zustände eingeführt werden.

Der Nachteil dieses Verfahrens ist allerdings, dass lediglich die Reihenfolge der Aktivitäten zur Laufzeit geändert werden kann. In manchen Situationen ist es aber erforderlich Aktivitäten einzufügen oder zu löschen. Dafür müsste man wieder auf komplexere Ad hoc-Änderungen zurückgreifen. Die Autoren weisen allerdings auch darauf hin, dass ihr Ansatz kein Ersatz für Ad hoc-Änderungen ist, sondern lediglich diese soweit wie möglich vermeiden soll.

Eine weitere generelle Schwierigkeit ist es, dass nicht alle Ausnahmen bei der Modellierung vorhergesehen werden können. Änderungen in der Reihenfolge, welche nicht vorab definiert wurden, sind durch diesen Ansatz leider nicht möglich. Außerdem müssen die Prioritäten im Vorfeld definiert werden. Es ist dem Nutzer nicht möglich, die Prioritäten dynamisch zur Laufzeit zu verändern. Dies stellt eine erheblich Einschränkung dar. Aus meiner Sicht ist es den Autoren ein vielversprechender Ansatz gelungen, den Grad der Flexibilität, ohne größere Erweiterungen des Workflow-Modells, zu erhöhen.

10. Probleme bei unternehmensübergreifenden Workflows

Häufig werden WFMS auch zur Koordination der Arbeit zwischen Unternehmen eingesetzt. Vor allem im Bereich des Supply Chain Managements. Die Unternehmen bieten dabei öffentliche Sichten auf ihre Prozesse an. Dabei sind nur bestimmte Aktivitäten nach außen hin sichtbar. Aktivitäten werden dabei gezielt verborgen, um den gesamten Geschäftsprozess nicht zu veröffentlichen. Auf diese Weise soll verhindert werden, dass ein Mitbewerber einen Einblick in die innerbetrieblichen Abläufe erhält. Auf der anderen Seite wird somit auch die Komplexität reduziert. Die Geschäftspartner müssen nur Schnittstellen gegen die Aktivitäten der öffentlichen Sicht zur Verfügung stellen.

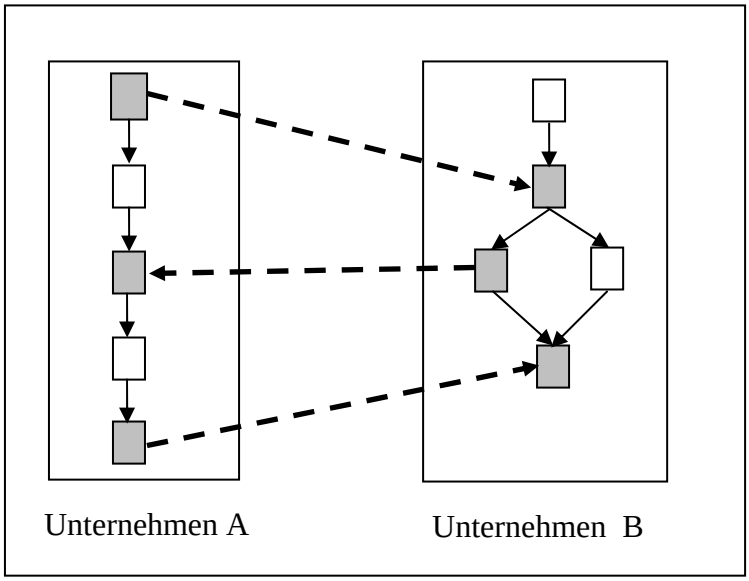

Abbildung 15 : Skizze von unternehmensübergreifenden Workflows

In Abbildung 15 ist ein Beispiel skizziert. Die grauen Aktivitäten stellen die öffentliche Sicht dar. Der Datenaustausch zwischen Unternehmen A und B findet

zwischen diesen Aktivitäten statt. Sowohl Unternehmen A als auch Unternehmen B verbergen Aktivitäten, weiß dargestellt, um die internen Abläufe geheim zu halten.

Die Sprache WS-BPEL bietet hierfür die Möglichkeit *Abstract Processes* zu definieren. Hierbei handelt es sich um einen Teilprozess eines internen Prozesses, der veröffentlich werden kann, ohne interne Details, die einen Wettbewerbsvorteil darstellen, zu veröffentlichen. Auf diese Weise kann das Protokoll (*Business Protocol*) spezifiziert werden, welches den Nachrichtenaustausch zwischen Unternehmen regelt [28].

Die Geschäftprozesse werden aber nun ständig modifiziert. Ein ständiger Abgleich mit den Geschäftspartnern ist allerdings nicht immer möglich oder sehr aufwändig, deshalb muss gewährleistet sein, dass die internen Änderungen zu keinen Inkonsistenzen mit den öffentlichen Sichten führen.

Eine Lösung für dieses Problem stellen die Ansätze dar, welche von Van der Aalst vorgeschlagen wurden [18]. Die öffentliche Sicht auf den Workflow stellt hier jeweils die Superklasse dar. Wenn nun die internen, nicht öffentlichen Workflows Unterklassen nach den beschriebenen Vererbungsregeln sind, ist eine Abbildung auf die öffentliche Sicht immer möglich. Dies wird dadurch gewährleistet, indem ausschließlich die Modifikationen benutzt werden, welche die Vererbungsrelation bewahren (siehe Kapitel 6). Aus Sicht eines externen Partners erscheint es dann so, als ob sich die Geschäftsprozesse des Partners gar nicht geändert hätten.

Meiner Meinung nach ist dieser Ansatz aber in der Praxis nicht ganz unproblematisch. Die Hauptproblematik ist, dass der Ansatz von Van der Aalst auf Petrinetzen basiert und Datenflussaspekte nur ungenügend betrachtet werden.

Allerdings ist der Van der Aalst Ansatz auch sehr weit gefasst. Die meisten Änderungen, die in der betrieblichen Praxis denkbar sind, z. B. Hinzufügen von zusätzlichen Aktivitäten oder Einführen von parallelen Zweigen, zerstören nicht die Vererbungsrelationen. Deshalb stellt der Ansatz von Van der Aalst, meiner Meinung nach, einen wesentlichen Fortschritt dar, da umfangreiche Modifikationsoperationen angeboten werden. Ein weiterer Vorteil ist, dass ein Formalismus angeboten wird, mit dem die Korrektheit der durchgeführten Modifikation stets einwandfrei bewiesen werden kann. Allerdings beschränkt sich dieser Ansatz lediglich auf Petrinetze, die, vor allem im Bereich des Datenflusses, erhebliche Schwächen haben, Geschäftsprozesse exakt zu beschreiben.

11. Vergleich und Anwendbarkeit auf WSBPEL-basierte Geschäftsprozesse

11.1 Vergleich

Die einzelnen Forschergruppen verfolgen sehr unterschiedliche Lösungsansätze, jeder einzelne hat seine individuellen Stärken und Schwächen. EinVergleich ist deshalb nicht ganz unproblematisch. Im folgenden sollen die eingesetzten Modelle und die Art der Modifikation betrachtet werden.

Autoren	Modell	Trennung Daten- und Kontrollfluss	Art der Modifikation
Dadam, Reichert	Graphenbasiert (ADEPT)	Ja	Sprache (ADEPT$_{flex}$)
Casati	Graphenbasiert	Ja	Sprache (WFML)
Ellis	Petrinetze	Nein	Austausch eines Teilgraphen
Sadiq	Petrinetze	Nein	Austausch eines Teilgraphen
Sadiq, Mangan	Dynamische Workflows	Nein	dynamíscher Aufbau
Van der Aalst	Petrinetze	Nein	Einfügen und Löschen von Teilgraphen

Tabelle 1 : Vergleich der Modelle

Bei petrinetzbasierten Modellen werden Änderungen als Austauschen eines Teilnetzes betrachet. Diese müssen separat generiert werden und auf strukturelle Korrektheit geprüft werden. Dies erfolgt durch umfangreiche Graphenanalysen, die in Polynomialzeit durchführbar sind. Bei Van der Aalst sind die Einschränkungen am größten. Teilnetze können hier nur an bestimmten Stellen eingefügt oder gelöscht werden. Bei graphenbasierten Ansätze erfolgt die

Modifikation durch eine Änderungssprache. Diese besteht aus einer Menge von elemtaren Änderungsprimitiven, mit denen sich Graphenelemte einfügen, löschen oder verschieben lassen. Alle diese Sprachen sind vollständig, minimal und konsistent. Bei Änderungen auf der Instanzebene muss zusätzlich noch eine Menge von Regeln erfüllt sein, damit ein Änderungsprimitiv angewandt werden kann. Jedes dieser Änderungsprimitive ist mit einer Menge von Regeln verknüpft, die den aktuellen Zustand der laufenden Instanz berücksichtigen. Auf diese Weise sollen Inkonsistenzen vermieden werden. Dadam schlägt zusätzlich vor, dass dem Nutzer eine Änderungssprache auf hohem semantischen Niveau präsentiert werden soll. Diese sematisch hohen Änderungsoperationen sollen dann automatisch auf eine Sequenz von Änderungsprimitiven abgebildet werden. Auf diese Weise soll es weitgehend vermieden werden, dass ein Nutzer strukturelle Inkonsistenzen, wie z. B. Deadlocks, erzeugen kann. Außerdem soll die Akzeptanz und Benutzerfreundlichkeit, durch eine intuitive Änderungssprache, deutlich gesteigert werden. Einen völlig anderen Ansatz verfolgen Sadiq und Mangan mit den dynamischen Workflows, hier wird der aktuelle Zustand der laufenden Instanz berücksichtigt und dem Nutzer ein Menge von anwendbaren Operationen präsentiert.

Eine generelles Merkmal von Petrinetzen ist die mangelnde Trennung von Daten und Kontrollfluss. Sadiq und Mangan klammern Datenflussaspekte bei den dynamischen Workflows explizit aus. Auf diese Weise können viele Probleme umgangen werden, allerdings wird dadurch auch die Anwendbarkeit auf realistische Szenarien reduziert, da viele Änderungen in der Praxis lediglich den Datenfluss betreffen.

Autoren	Vorabdefinierte Ausnahmen	Behandlung von nicht verträglichen Instanzen	Gleichzeitige Modifikation von Schema und Instanz
Dadam, Reichert	ja	ECA – Tabelle	Überlappungserkennung
Casati	nein	Abort, Flush, hybrider Graph (manuell)	nicht möglich
Ellis	nein	immediate change, hybrider Graph (SCOC)	möglich
Sadiq	nein	Abort, Flush, Compliance Graph (partielles Abort)	nicht möglich
Sadiq, Mangan	nein	unverträgliche Instanzen nicht möglich	kein Schema vorhanden
Van der Aalst	nein	unverträgliche Instanzen nicht möglich	möglich

Tabelle 2 : Ansätze für Ad hoc – Modifikationen

Der Ansatz von Dadam und Reichert bietet als einziger die Möglichkeit Ausnahmen bereits im Vorfeld zu definieren. Auf diese Weise wird der Grad der Flexibilität erhöht und Ad hoc-Modifikationen zur Laufzeit können teilweise vermieden werden.

Der Verträglichkeitsbegriff ist bei allen Autoren sehr ähnlich. Die petrinetz-basierten Verfahren definieren die Verträglichkeit über die Ausführungssequenz, wohingegen bei den graphenbasierten Verfahren auf der Ausführungshistorie aufgebaut wird. Die Idee, welche dahinter steckt, ist allerdings die gleiche. Wenn das bisherige Verhalten der laufende Instanz auch mit dem geänderten Schema erzeugt werden kann, dann ist sie mit dem geänderten Schema verträglich.

Bei der Behandlung von nicht verträglichen Instanzen werden unterschiedliche Vorschläge gemacht. Der Ansatz von Van der Aalst und der Ansatz der dynamischen Workflows von Sadiq und Mangan umgehen diese Problematik. Bei van der Aalst wird die Migrierbarkeit durch spezielle Modifikationsoperationen gewährleistet, d.h. dass durch Modifikationen keine unverträglichen Instanzen erzeugt werden können. Wohingegen bei flexiblen Workflows keine Schemata vorhanden sind und deshalb eine Migration niemals erforderlich sein wird (siehe Kapitel 8). Dadam und Reichert schlagen eine ECA – Tabelle vor. Hier werden die laufenden Instanzen eingetragen, die sich gerade in einem Schleifendurchlauf befinden, sofern dieser beendet wurde, wird die Migration durchgeführt. Casati, Ellis und Sadiq machen ähnliche Vorschläge. Die unverträglichen Instanzen sollen abgebrochen oder nach dem alten Schema zu Ende laufen oder auf einen hybriden Graphen migriert werden. Bei Casati muss dieser manuell erstellt worden sein, wohingegen dieser bei Ellis und Sadiq automatisch erzeugt werden soll. Die Idee hinter dem Compliance Graphen von Sadiq ist, dass Kompensationsaktivitäten eingefügt werden, und so das Petrinetz in einen verträglichen Zustand zurückversetzt wird. Bei Ellis wird mittels eines Synthetic Cut-Over Change die Old Region mit der New Region überlagert (siehe Kapitel 4).

Gleichzeitige Änderungen an dem Schema und einer abgeleiteten Instanz werden bei dem Ansatz von Dadam (siehe Kapitel 5), sowie dem Ansatz von Ellis, als auch dem Ansatz von Van der Aalst unterstützt. Bei dem Ansatz von Ellis ist es unbedeutend, ob eine Instanz zusätzlich zum Schema geändert wurde. Durch das SCOC lassen sich immer kompatible hybride Graphen erzeugen. Bei Van der Aalst ist die Verträglichkeit immer garantiert, sofern nur die vorgeschlagenen Änderungsoperationen angewandt werden, dabei ist es nicht von Belang, ob diese auf Instanz- oder Schemaebene angewandt wurden.

Autoren	Unterstützte Änderungoperationen	Ansätze für unternehmensübergreifende Anwendungen	Ansätze zur verteilten Implementierung
Dadam, Reichert	additiv, subtraktiv, order changing	Nein	ADEPT*Distribution*
Casati	additiv, subtraktiv, order changing	Nein	Nein
Ellis	additiv, subtraktiv, order changing	Nein	Nein
Sadiq	additiv, subtraktiv, order changing	Nein	Nein
Sadiq, Mangan	additiv	Nein	Nein
Van der Aalst	additiv, subtraktiv	Ja	Nein

Tabelle 3 : Unterstützte Änderungsoperationen

Es gibt drei Arten von Änderungsoperationen, die behandelt wurden : Einfüge- (additiv) und Löschoperationen (subtraktiv), sowie Änderungen in der Abfolge der Aktivitäten (order changing). Die ersten beiden Arten von Änderungen können sowohl den Kontroll- als auch den Datenfluss verändern. Bei dem Ansatz der flexiblen Workflows von Sadiq und Mangan werden lediglich additive Änderungen unterstützt. Der Ansatz von Van der Aalst unterstützt Order changing - Änderungen nur eingeschränkt. Diese können zwar realisiert werden, indem ein Teilnetz gelöscht und an anderer Stelle wieder eingesetzt wird. Allerdings ist dies, aufgrund der speziellen Änderungsoperationen, nicht immer möglich.

Der Ansatz von Van der Aalst bietet als einziger einen Ansatz um Änderungen zu realisieren, die Sichten auf den Workflow erhalten (siehe Kapitel 10).

Mit ADEPT*Distribution* machen Dadam und Reichert als einzige Vorschläge, wie Ad hoc - Änderungen in verteilten Systemen implementiert werden können.

11.2 Anwendbarkeit dieser Konzepte auf WS-BPEL-basierte Business-Prozesse

Aus meiner Sicht sind alle petrinetzbasierten Verfahren nur sehr eingeschränkt auf Geschäftsprozesse anwendbar, welche mit WS-BPEL erzeugt wurden. Das Hauptproblem besteht in der mangelnden Trennung von Kontroll- und Datenfluss. Änderungen, die lediglich den Datenfluss betreffen, können mit diesen Konzepten nicht erfasst werden. Außerdem sind in diesen Verfahren keine Kompensationsaktivitäten berücksichtigt. Alle petrinetzbasierten Verfahren basieren auf der Annahme, dass die Petrinetze jederzeit und unmittelbar in frühere Zustände zurückversetzt werden können. Dies ist aber bei anwendungsnahen Workflows nicht immer der Fall. Hier kann es vorkommen, dass Aktivitäten gar nicht oder nur mit sehr großem Aufwand rückgängig gemacht werden können. Die Problematik, dass eventuell Dateninkonsistenzen durch einen Rücksprung erzeugt werden können, wird bei Petrinetzen komplett ausgeklammert. Für einen anwendungsnahen Einsatz muss diese Problematik jedoch unbedingt gelöst werden.

Die Ansätze von Dadam, sowie die von Casati erscheinen mir am besten geeignet für eine Anwendung auf WSBPEL – basierte Prozesse. Das Modell, welches diesen Ansätzen zu grunde liegt, ist dem von WSBPEL relativ ähnlich. Vor allem ADEPT bietet einen blockorientierten Ansatz, der relativ ähnlich ist. Außerdem ist das Konzept der success dependencies, welches in ADEPT zum Einsatz kommt, dem Konzept der spheres of control von WSBPEL stark nachempfunden. Aufgrund dieser Ähnlichkeiten sind die präsentierten Ansätze relativ problemlos auf WSBPEL übertragbar. Änderungen auf Instanzebene können relativ einfach implementiert werden. Effiziente Algorithmen, die eine Konsistenzprüfung und

anschließend eine Neubewertung der Aktivitätszustände durchführen, sind bereits vorhanden (siehe Kapitel 3). Was die verteilte Implementierung angeht, wurden mit ADEPT$_{Distribution}$ schon sehr ausgereifte Konzepte entwickelt, die sich aus meiner Sicht ohne größere Probleme auf WSBPEL übertragen lassen (siehe Kapitel 7). Allerdings hat ADEPT in manchen Punkten auch Schwächen, deshalb schlage ich vor, einige Ansätze der anderen Forschergruppen zu übernehmen.

Für die Behandlung von nicht verträglichen Instanzen schlage ich den Ansatz vor, diese in Klassen einzuteilen (vgl. Casati, Ellis, Sadiq). Als Kriterium hierfür kann der Fortschritt der laufenden Instanz herangezogen werden. Für jede dieser Klasse muss nun ein geeignetes Vorgehen bestimmt werden. Als Möglichkeiten stehen hier ein Abort, Flush oder die Migration auf einen hybriden Graphen zur Verfügung. Beispielsweise könnten Instanzen in einem sehr frühen Stadium abgebrochen werden (Abort), wohingegen Instanzen, welche weit fortgeschritten sind, nach dem alten Schema zu Ende laufen könnten (Flush). Der hybride Graph muss manuell erstellt werden. Der Ansatz von Ellis, diesen automatisch zu erstellen, halte ich für noch nicht anwendbar. Das Hautpproblem ist es, die Change Region effizient automatisch zu bestimmen. Der Compliance Graph, den Sadiq vorschlägt, wird in der Regel suboptimal[10] sein. Allerdings kann dieser automatisch generiert werden. Deshalb bietet sich die Möglichkeit an, ihn dem Nutzer als Vorschlag zu präsentieren. Auf diese Weise könnte der Nutzer unterstützt werden.

Ein weitere wichtiger Punkt von WSBPEL sind die Abstract Processes. Van der Aalst bietet als einziger einen Ansatz, der Ad hoc - Modifikationen ermöglich und gleichzeitig die Konsistenz der öffentlichen Sicht auf den Prozess gewährleistet. Allerdings ist dieser Ansatz aufgrund des unterschiedlichen Modells (Petrinetz) nicht unverändert auf WSBPEL übertragbar. Aus meiner Sicht müssten hierzu die Vorbedingungen einer Änderungsoperation erweitert werden. Wenn ein Nutzer beispielsweise eine Aktivität löschen möchte, muss zusätzlich zum aktuellen Zustand der Instanz noch überprüft werden, ob diese Aktivität ein Bestandteil des Abstract Process ist. Wenn dies zutrifft, dann muss die Löschung verweigert werden.

Gleichzeitige Änderungen auf der Instanz- und der Schemaebene sind meiner Meinung nach noch nicht implementierbar. Dadam, Reichert, Rinderle haben schon erste Schritte unternommen, dies zu realisieren, allerdings sind noch zu viele Probleme ungelöst (siehe Kapitel 5).

[10] Da hier eventuell mehr Aktivitäten rückgängig gemacht werden, als erforderlich. Diese Vermutung muss aber in einem Praxistest noch näher untersucht werden.

12. Fazit und Ausblick

Aus meiner Sicht stellen Ad hoc-Änderungen eine große Erweiterungsmöglichkeit bestehender WFMS dar. Es können völlig neue Anwendungsfelder erschlossen werden. Vor allem Bereiche in denen bisherige WFMS zu starr und unflexibel waren. Als Beispiel lässt sich hier der Einsatz in einem Krankenhaus nennen. Außerdem wird es Unternehmen ermöglicht, schneller und flexibler auf Kundenwünsche und Änderungen des Marktes zu reagieren. Fehler im Arbeitsablauf können durch Ad hoc-Änderungen ebenfalls sehr viel schneller und einfacher behandelt werden. Durch eine Ad hoc-Änderung können nun, relativ einfach, alternative Aktivitäten eingefügt werden. Auf diese Weise kann sich ein Unternehmen erhebliche Wettbewerbsvorteile gegenüber seinen Mittbewerbern sichern. Aus diesen Gründen sollte die Möglichkeit für Ad hoc-Änderungen in bestehende WFMS und Workflowmodelle unbedingt integriert werden.

13. Quellenverzeichnis

[1] M. Reichert, P. Dadam, T. Bauer: *Dealing With Forward and Backward Jumps in Workflow Management Systems.* Informatik Forsch. Entw., 18(3-4):132-151, 2004 (Invited reprint of an earlier Springer paper; the original publication is available on DOI 10.1007/s00450-004-0157-5 at http://link.springer.de)

[2] S. Rinderle, M. Reichert, P. Dadam: *Supporting Workflow Schema Evolution By Efficient Compliance Checks*. Technical Report UIB-2003-02, University of Ulm, Faculty of Computer Science, May 2003

[3] S. Rinderle, M. Reichert, P. Dadam: *Effiziente Verträglichkeitsprüfung und automatische Migration von Workflow-Instanzen bei der Evolution von Workflow-Schemata.* Informatik Forschung und Entwicklung, 17(4):177-97, December 2002 (The original publication is available on DOI 10.1007/s00450-002-0122-0 at http://link.springer.de).

[4] M. Reichert, T. Bauer, T. Fries, P. Dadam: *Modellierung planbarer Abweichungen in Workflow-Management-Systemen.* Proc. GI-Arbeitskonferenz Modellierung‘02, Tutzing/Germany, March 2002, pp. 183-194 (GI Lecture Notes in Informatics, Band P-12)

[5] T. Bauer, M. Reichert, P. Dadam: *Adaptives und verteiltes Workflow-Management.* Proc. Datenbanksysteme in Büro, Technik und Wissenschaft (BTW‘2001), Oldenburg, March 2001, pp. 47-66 (Best Paper)

[6] M. Reichert: Prozessmanagement im Krankenhaus - Nutzen, Anforderungen und Visionen. das Krankenhaus, 92(11):903-909, November 2000 (Invited Paper)

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

[8] S. Rinderle, M. Reichert, P. Dadam: *On Dealing With Structural Conflicts Between Process Type and Instance Changes*, 2nd Int'l Conf. Business Process Management (BPM'04), Potsdam, Germany, June 2004, LNCS 3080, pp.274-289.

[9] S. Rinderle, M. Reichert, P. Dadam: *Disjoint and Overlapping Process Changes - Challenges, Solutions, Applications.* Proc. 12th Int'l Conf. Cooperative Information Systems (CoopIS'04), Larnaca, Cyprus, October 2004 (accepted for publication)

[10] S. Rinderle, M. Reichert, P. Dadam: *Correctness Criteria For Dynamic Changes in Workflow Systems - A Survey*. Data and Knowledge Engineering, Special Issue on Advances in Business Process Management, 50(1):9-34 (2004)

[11] M. Reichert, S. Rinderle, U. Kreher, P. Dadam: *Adaptive Process Management with ADEPT2*. 21th Int'l Conf. on Data Engineering (ICDE'05), Tokyo (Demo Session)

[12] Fabio Casati and Stefano Ceri and Barbara Pernici and Giuseppe Pozzi : *Workflow Evolution.* International Conference on Conceptual Modeling / the Entity Relationship Approach, 438-455 (1996)

[13] C.A. Ellis, K. Keddara, and G. Rozenberg. *Dynamic Change within Workflow Systems.* In N. Comstock,C.A. Ellis, R. Kling, J. Mylopoulos, and S. Kaplan, editors, Conference on Organizational Computing Systems, Proceedings, pages 10 – 21, Milpitas, California, August 1995. ACM Press, New York, 1995.

[14] S. Shazia and S. Olivera and M. Maria and E. Orlowska. *Managing Change and Time in Dynamic Workflow Processes.* SMO99 Shazia Sadiq, Olivera Marjanovic, Maria E. Orlowska (1999) ManagingChange and Time in Dynamic Workflow Processes. (To Appear) International Journal of Cooperative Information Systems (1999).
http://citeseer.ist.psu.edu/olivera99managing.html

[15] Peter Mangan and Shazia Sadig. *On Building Workflow Models for Flexible Processes.* Thirteenth Australasian Database Conference (ADC2002). Melbourne, Australia (2002).
http://citeseer.ist.psu.edu/mangan02building.html

[16] Shazia Sadiq (1999) *Workflows in Dynamic Environments – Can they be managed?* Proceedings of The Second International Symposium on Cooperative Database Systems for Advanced Applications (CODAS99), Woollongong, Australia. March 27-28, 1999.

[17] G.Teege: *Flexible Workflows: Mitgestaltung durch die Ausführenden.* Proc. Workshop Flexibilität und Kooperation in Workflow-Management-Systemen, 1998, S. 13-21

[18] v.d. Aalst, w.,Basten, T. : *Inheritance of workflows: An approach to tackling problems related to change*. theoret Comp. science 270 (2002) 125-203

[19] W.M.P. van der Aalst, K.M. van Hee, and G.J. Houben. *Modelling workflow management systems with high-level Petri nets.* In G. De Michelis, C. Ellis, and G. Memmi, editors, Proceedings of the second Workshop on Computer-Supported Cooperative Work, Petri nets and related formalisms, pages 31-50, 1994.

[20] W.M.P. van der Aalst. *Making Work Flow: On the Design, Analysis and Enactment of Business Processes* (inaugural lecture given at 30 November 2001).
Eindhoven University of Technology, Eindhoven, The Netherlands, 2001.
http://is.tm.tue.nl/staff/wvdaalst/publications/publications.htm

[21] IBM Systems Journal Artikel *"Business process choreography in WebSphere: Combining the power of BPEL and J2EE"*
http://www.research.ibm.com/journal/sj43-2.html

[22] T.M. Koulopoulos. *The Workflow Imperative*. Van Nostrand Reinhold, New York, 1995.

[23] W.M.P. van der Aalst.
How to Handle Dynamic Change and Capture Management Information: An Approach Based on Generic Workflow Models.
International Journal of Computer Systems, Science, and Engineering, 16(5):295-318, 2001.

[24] P. Dadam, M. Reichert, K. Kuhn: *Clinical Workflows - The Killer Application for Process-oriented Information Systems?* Proc. 4th Int'l Conf. on Business Information Systems (BIS'2000), Poznan, Poland, April 2000, pp. 36-59

[25] **Brunner, Jörg :** *Ad hoc Modifikation von Workflows unter besonderer Betrachtung des Workflow Management Systems IBM MQ Series Workflow*
Stuttgart, Univ., Fakultät Informatik, Diplomarbeit Nr. 1815 Stuttgart, 2000

[26] Namics Whitepaper: *Microsoft Sharepoint Solution: Workflow Management*
www.namics.com/fileadmin/user_upload/pdf/SPS_Workflow_Whitepaper_d_Juli_04_v1.6.pdf

[27] *An Introduction to Workflow*
by Charles Plesums (Fellow, USA) extracted from the Workflow Handbook 2002
http://www.wfmc.org/information/introduction_to_workflow02.pdf

[28] Business Process Execution Language for Web Services Version 1.1
http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/

Erklärung

Hiermit versichere ich, die Arbeit selbstständig verfasst und nur die angegebenen Quellen verwendet zu haben.

Unterschrift :

Böblingen, den 28. September 2005