Universität Ulm

Fakultät für Informatik

Abteilung Datenbanken und Informationssysteme

Effiziente Realisierung von Prozess–Schemaevolution in Hochleistungs–Prozess–Management–Systemen

Diplomarbeit

vorgelegt von
Markus Lauer

17. Dezember 2004

1. Gutachter: Prof. Dr. Peter Dadam
2. Gutachter: Dr. Manfred Reichert

Zusammenfassung

Immer mehr Unternehmen setzen zur Verwaltung und Überwachung ihrer betrieblichen Prozesse sogenannte Prozess-Management-Systeme (PMS) ein. Eine der wichtigsten Voraussetzungen für einen sinnvollen Einsatz ist es, dass PMS die Eigenschaft der Adaptivität besitzen: Sie müssen Änderungen der hinterlegten Prozessvorlagen und der sich in Ausführung befindlichen Prozesse zur Laufzeit zulassen, damit die Unternehmen in der Lage sind, flexibel und schnell auf neue Anforderungen zu reagieren. Dies stellt jedoch hohe Ansprüche an ein PMS: Effiziente Algorithmen müssen dafür sorgen, dass die Änderungen nicht zu inkonsistenten Zuständen von Prozessvorlagen und -instanzen führen. Modifikationen der Prozessvorlagen müssen, wo vom Ausführungsfortschritt her möglich, auf die darauf basierenden Prozessinstanzen propagiert werden. Bei Instanzen, die sich aufgrund von Ad-hoc-Änderungen in ihrem Ablauf gegenüber der Vorlage unterscheiden, müssen zusätzlich ohne großes Eingreifen seitens der Benutzer Probleme erkannt und beseitigt werden, die sich bei der Migration auf die neue Version aus überlappenden oder widersprüchlichen Instanz- und Vorlagenänderungen ergeben. Dabei dürfen die anderen, parallel ablaufenden Funktionen des Systems nicht beeinträchtigt werden, selbst wenn in realen Anwendungen tausende von Instanzen migriert werden müssen. Hinzu kommt, dass dem System für diese Aufgaben nur eingeschränkt Ressourcen, wie z. B. Speicher, zur Verfügung stehen. Alle diese Anforderungen verlangen nach einer flexiblen und ressourcensparenenden Architektur sowie nach einer effizienten Implementierung. Die auf dem Markt erhältlichen Produkte bieten entweder gar keine oder nur eingeschränkte Änderungsmöglichkeiten zur Laufzeit oder erfüllen die angesprochenen Anforderungen nur unzureichend. Wir entwickeln in unserem Projekt AristaFlow ein Prozess-Management-System, welches Änderungen zur Laufzeit vollständig unterstützt.

Die Diplomarbeit stellt dafür einen Ansatz zur internen Repräsentation von Prozessvorlagen und -instanzen vor, der volle Adaptivität bereitstellt und den Anforderungen an einen Praxiseinsatz genügt. Darauf aufbauend werden zwei Optimierungsansätze diskutiert, die das Ziel haben, den Migrationsprozess im Ganzen zu beschleunigen. Bei einem Prozess-Management-System werden viele Aktionen gleichzeitig ausgeführt. Ohne Kontrolle bzw. Synchronisation durch das System kann es dabei zu Problemen aufgrund von konkurrierenden Aktionen kommen. Diese Arbeit analysiert die Probleme, die bei der vorgestellten Architektur zwischen der Schemaevolution, der Ad-hoc-Modifikation, der Migration und dem Weiterschalten von Instanzen auftreten können und zeigt Lösungsmöglichkeiten auf.

Vorwort

Meine Diplomarbeit durfte ich über ein sehr interessantes, aber auch sehr weitgreifendes Thema verfassen.

An dieser Stelle möchte ich mich bei Herrn Prof. Dr. Peter Dadam bedanken, dass ich mich in seiner Abteilung in dieses Metier einarbeiten durfte und so eine solide sachliche Grundlage für diese Arbeit bekam.

Ganz besonders danke ich meinen Betreuern Frau Stefanie Rinderle und Herrn Dr. Manfred Reichert, die mir immer mit Rat und Tat beiseite standen, stets ein offenes Ohr für meine Fragen hatten und mir mit ihren Anregungen immer wieder neue Impulse gaben.

Besonderer Dank gilt auch meinen Eltern, meiner Schwester und besonders meinen Studienkollegen, die mich immer – mehr oder weniger freiwillig – unterstützten, indem sie Korrektur gelesen, mein ständiges Klagen erduldet oder mir Mut zugesprochen haben. Besonders nett fand ich von meinen Studienkollegen, dass sie aus Solidarität den Sommer über mit mir in der Universität saßen und arbeiteten, anstatt die Tage im Urlaub am See oder am Meer zu verbringen.

Namentlich erwähnen möchte ich noch Ulrich Kreher, der mir viel geholfen hat, gerade die anstrengende, nervenaufreibende Schlussphase gut zu überstehen, auch wenn seine konstruktive Kritik mir so manchen Schock versetzt hat und einen riesen Berg Arbeit vor meinem geistigen Auge entstehen ließ. Auch meinen Kollegen Elmar Schoch möchte ich hier nicht vergessen.

All denen, die mich so super unterstützt haben und welchen ich mit meinem dünnen Nervenkostüm zu schaffen machte, möchte ich für ihre Nachsicht und Geduld einen Anteil an dieser Arbeit zusprechen.

Ulm, den 17. Dezember 2004

Inhaltsverzeichnis

Kapitel 1

Einführung

„Wer rastet, der rostet!“ lautet ein weit verbreitetes Sprichwort. Mit diesem Sprichwort lässt sich sehr gut die Situation der Unternehmen in der heutigen globalen Geschäftswelt beschreiben. Wer es sich heute erlaubt, sich auf dem bisher erreichten Niveau bezüglich angebotener Produkte bzw. Dienstleistungen, Qualität und Kosten auszuruhen, wird sehr schnell überrollt werden von der Masse an Konkurrenten. Um im globalen Wettbewerb bestehen zu können, ist es unbedingt notwendig, sich ständig und schnell auf einen sich verändernden Markt einzustellen, flexibel auf neue Anforderungen zu reagieren, sowie die Entwicklungs- und Produktionszeiten zu verkürzen. Gleichzeitig müssen die Gesamtkosten minimiert und die Qualität der Arbeit maximiert werden. Die Qualität der Arbeit umfasst dabei nicht nur die Güte der Produkte bzw. der Dienstleistungen, sondern auch, dass die Erfüllung des Kundenwunsches in der geforderten Zeit erfolgt.

Ein wichtiger Schritt, um die Kosten zu minimieren und die Qualität zu maximieren, ist es, die Geschäftsprozesse des Betriebes als Ganzes zu erfassen, auf Schwachstellen, Engpässe sowie unnötige Arbeitsschritte zu analysieren, den Prozess auf diese Erkenntnisse hin zu optimieren und dann die Einhaltung der Arbeitsabfolgen und der vorgegebenen Zeiten zu überwachen (vgl. [RD02, S. 1-4]).

Stand heute ist, dass in den Unternehmen zwar die einzelnen Prozessschritte für sich betrachtet höchst effizient ausgeführt werden, der Prozess als Ganzes aber nicht an das Optimum herankommt. Erhöhte Produktionszeiten sind die Folge. Der Grund für die fehlende Optimierung auf Prozessebene ist in der Arbeitsteilung zu suchen. Ein Mensch versucht von Natur aus, möglichst wenig Energie für seine Arbeit aufzubringen, und optimiert deshalb seinen Arbeitsablauf bewusst oder unbewusst. Da eine Person aufgrund der (notwendigen) Arbeitsteilung nicht mehr für den gesamten (Herstellungs-)prozess (z. B. eines Autos) sondern nur für einzelne Teilschritte (z. B. Innenverkleidung anbringen) verantwortlich ist, werden auch nur die einzelnen Teilschritte für sich betrachtet im Ablauf optimiert. Das Zusammenspiel der einzelnen Teilprozesse bleibt von der Optimierung aber unberührt (vgl. [RD02, S. 1-2 ff.]).

Der Gesamtprozess jedoch lässt sich z. B. dahingehend verbessern, dass einzelne Teilschritte, wo möglich, statt bisher sequentiell parallel ausgeführt werden [RD02, S. 4-5]. Damit kann die Produktionszeit erheblich verkürzt werden. Oder es lassen sich zwei getrennt ausgeführte Arbeitsschritte zusammenfassen [RD02, S. 4-5]. Damit lässt sich Zeit und Geld für Transport von einem Arbeitsplatz zu einem anderen einsparen. Der freigewordene Arbeitsplatz kann dann anderweitig genutzt werden. Identifizierte Engpässe können durch die Einführung von zusätzlichen parallelen Arbeitsschritten beseitigt werden.

Um überhaupt Ansatzpunkte für Optimierungen finden zu können, muss ein Überblick über die teilweise sehr großen und komplexen Arbeitsabläufe eines Betriebes geschaffen werden. Prozessmodellierungstools, wie z. B. ARIS Toolset[1], können dabei helfen. Diese Tools erlauben es, Ablaufstrukturen graphisch zu modellieren und darauf basierend Simulationen durchzuführen [RD02, S. 3-119 ff., S. 4-6]. Durch diese Simulationen können z. B. Engpässe erkannt werden, oder es kann damit überprüft werden, ob Änderungen am Ablauf die gewünschte Wirkung erzielen.

Doch was nützt der beste Arbeitsplan, wenn er in der Praxis nicht eingehalten wird, sei es, dass Arbeitsschritte durch den menschlichen Faktor vergessen werden (z. B. eine wichtige Unterschrift besorgen), oder dass die angesetzte Zeit nicht eingehalten wird, da z. B. Akten nicht aufgefunden werden? Unnötige Arbeit, Überschreitung von Fristen oder sogar Abbrüche von ganzen Vorgängen drohen. Zum Beispiel darf ein Arzt nicht operieren, wenn er keine Einverständniserklärung des Patienten vorweisen kann. Die entstandenen zusätzlichen Kosten können immens sein, besonders wenn der Betrieb auf Nicht-Erfüllung eines Vertrages verklagt wird. Hier kommen nun Prozess-Management-Systeme (PMS) ins Spiel.

1.1 Aufgaben von Prozess-Management-Systemen

PMS sind komplexe Softwaresysteme, die ein Unternehmen bei der Prozessabwicklung in Bezug auf Einhaltung der Arbeitspläne unterstützen sollen (vgl. [RD02, S. 1-37 ff.]): In den Systemen werden die optimierten Arbeitspläne als Prozessvorlagen hinterlegt. Anhand der Arbeitspläne und anhand der Informationen, wie weit ein Prozess fortgeschritten ist, bestimmen sie die als nächstes auszuführenden Arbeitsschritte und stellen den dafür in Frage kommenden Mitarbeitern entsprechende Arbeitsvermerke in deren persönliche Arbeitsliste. Zusätzlich stellen sie dem Bearbeiter, der die Aufgabe übernimmt, alle zur Bearbeitung notwendigen Daten zur Verfügung. Dadurch können z. B. Verzögerungen aufgrund nicht gefundener Akten vermieden werden. Dazu müssen alle für einen Prozessablauf relevanten Daten im System hinterlegt werden. Gleichzeitig überwachen die Systeme, dass sämtliche Aktivitäten in der vorgegebenen Zeitspanne in Angriff genommen und auch in der eingeplanten Zeit abgeschlossen werden. Drohen Zeitüberschreitungen, so werden die verantwortlichen Personen darauf aufmerksam gemacht.

PMS unterstützen aber nicht nur Aktivitäten, die von Menschenhand ausgeführt werden müssen, sondern auch Schritte, die ganz automatisch von einem System erledigt werden können. Zum Bei-

[1] http://www.ids-scheer.de/

spiel könnte ein PMS automatisch ein Buchungsprogramm veranlassen, den Betrag einer fälligen Rechnung auf das Konto des Gläubigers zu überweisen. Die Vorgehensweise des PMS bei automatischen Agenten ist ähnlich der bei menschlichen. Beim Starten der Aktivität „Rechnungsbetrag überweisen“ im Geschäftsprozess „Rechnung bezahlen“ ruft das PMS die entsprechende Software auf und übergibt die benötigten Daten, wie z. B. die Kontonummer und Bankleitzahl des Empfängers und den Rechnungsbetrag. Die Buchungssoftware führt die Transaktion aus und meldet durch einen entsprechenden Rückgabewert die erfolgreiche Überweisung. Das PMS nimmt diesen Wert entgegen und speichert ihn, um z. B. in einem späteren Schritt eine Buchungsbestätigung ausdrucken zu können. Danach bestimmt es die nächsten abzuarbeitenden Aufgaben.

Die Fähigkeit, auch automatische Agenten zu unterstützen, macht die PMS interessant für den Einsatz in einer dienstbasierten Architektur (englisch: *Service Oriented Architecture (SOA)*). Die Idee hinter einer dienstbasierten Architektur ist, mehrere unabhängige (Software-)dienste (Services), die jeder für sich eine bestimmte Aufgabe erfüllen, zusammenzuschalten und zu einer neuen Anwendung mit eigener Funktionalität zu kombinieren, wie in Abbildung 1.1 illustriert [RD02, S. 6-15 ff.]. Für die Steuerung der Abfolge der Service-Aufrufe (*Service Flow*) können

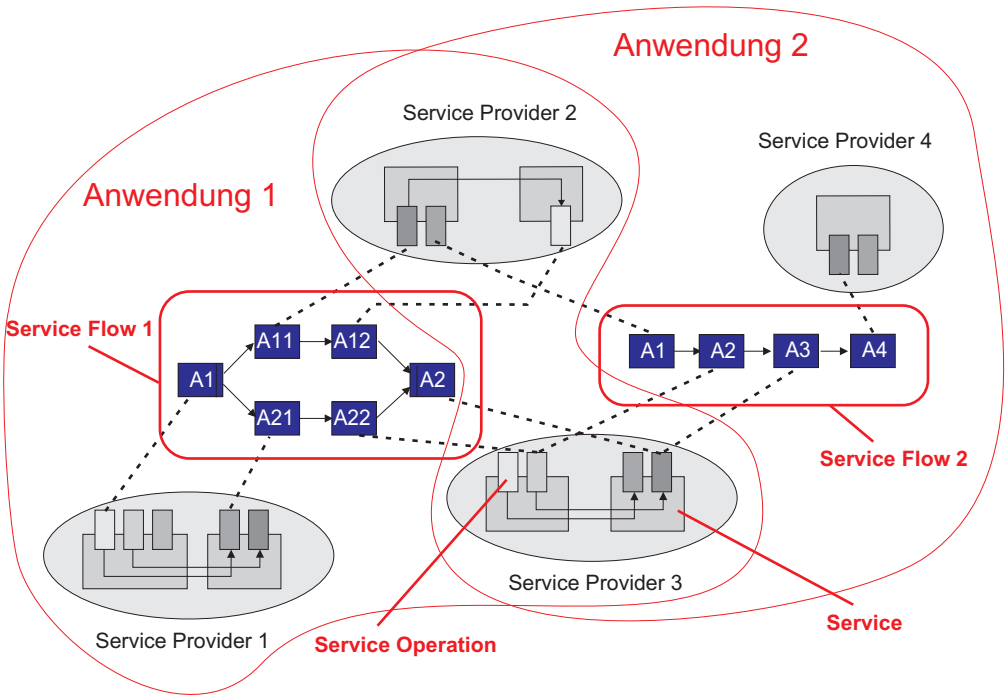

Abbildung 1.1: Service Flows und Services (in Anlehnung an [RD02, Abb. 6-13])

PMS eingesetzt werden. Das PMS bestimmt anhand der z. B. mit BPEL4WS[2] beschriebenen Prozessvorlagen die als nächstes auszuführenden Services, ruft diese gemäß den vorgegebenen Protokollen auf, übergibt die benötigten Parameter und nimmt nach erfolgter Abarbeitung die Ergebnisse entgegen, um sie den folgenden Diensten als Eingabe bereitzustellen. Der SOA kommt in der Software-Entwicklung eine stark zunehmende Bedeutung zu, da dieser Ansatz immense Kosteneinsparungen verspricht: Die Wiederverwendung bereits bestehender und im Einsatz erprobter Dienste verkürzt die Entwicklungs- und Programmierzeit – das Rad muss nicht mehr neu erfunden werden – und erspart aufwändige und teure Funktionstests von Eigenentwicklungen. Von der Beliebtheit des neuen Ansatzes zeugt der große Erfolg des bekanntesten Vertreters der SOA, die Web-Services. Und es zeigt, dass der Prozessgedanke – das nacheinander Abarbeiten von Arbeitsschritten bzw. Diensten – wieder im Vordergrund steht.

1.2 Adaptivität

In einem Unternehmen müssen viele verschiedene Geschäftsprozesse unterstützt werden. Für jeden unterstützten Typ wird in der Prozessvorlage das entsprechende Prozessschema im PMS hinterlegt. Das Prozessschema gibt den prinzipiellen Ablauf des Geschäftsprozesses wieder.

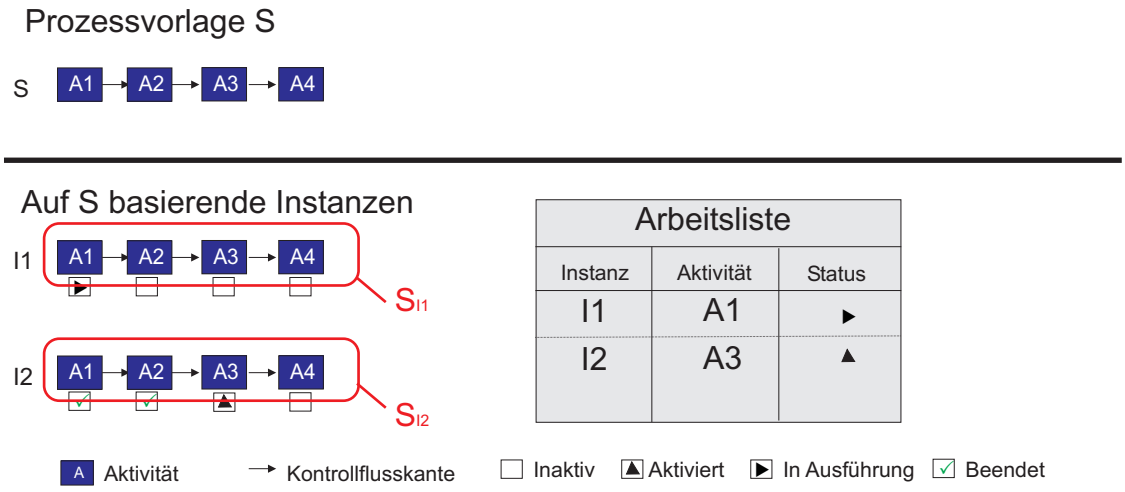

Abbildung 1.2: Das Schema S und darauf basierende Instanzen

Das in Abbildung 1.2 aufgeführte Schema S bestimmt z. B., dass die Aktivitäten A1, A2, A3 und A4 in dieser Reihenfolge sequentiell nacheinander ausgeführt werden müssen. Basierend auf der Prozessvorlage können Instanzen erzeugt und gestartet werden. Eine Prozessinstanz ist eine konkrete Ausprägung eines Prozesstyps. Jedem Prozesslauf ist eine Instanz zugeordnet. Die

[2]Business Process Execution Language for Web Services (http://www.ibm.com/developerworks/library/ws-bpel/)

Instanz gibt den Bearbeitungszustand der einzelnen Aktivitäten wieder. In vielen Umgebungen, wie z. B. in einer Klinik, können viele tausend Instanzen eines Prozesstyps zur gleichen Zeit aktiv sein, wobei sich jede in einem anderen Zustand befinden kann. Die Instanzen I1 und I2 aus Abbildung 1.2 basieren auf S, d. h. deren Laufzeitschemata S_{I1} bzw. S_{I2} stimmen mit dem Schema S der Vorlage überein. Die beiden Instanzen sind unterschiedlich weit fortgeschritten: Bei I1 befindet sich die erste Aktivität A1 in Ausführung. Damit steht I1 am Anfang der Prozessausführung. Bei I2 wurden dagegen bereits die Aktivitäten A1 und A2 vollständig und erfolgreich abgearbeitet. Laut S_{I2} steht damit die Aktivität A3 zur Ausführung an. Sie befindet sich im Zustand „AKTIVIERT“. Der zugeordnete Bearbeiter[3] hat einen entsprechenden Eintrag in seine Arbeitsliste gestellt bekommen.

Allerdings ist ein Geschäftsprozess nicht auf ewig in seinem einmal aufgestellten Ablauf festgelegt. Der Ablauf kann sich ändern. Gründe dafür gibt es viele [CCPP98, Aa01, AJ00]: Im Laufe des alltäglichen Betriebes können neue Optimierungsmöglichkeiten im Arbeitsablauf erkannt werden. Diese zukünftig zu berücksichtigen bedeutet, den Geschäftsprozess anzupassen. Auch werden häufig Fehler oder Unvollständigkeiten in der Prozessmodellierung erst im laufenden Betrieb festgestellt. Die Folge ist, dass Prozesskorrekturen durchgeführt werden müssen. Betriebliche Umstrukturierungen, wie das Auslagern von Produktionsabschnitten an externe Firmen (Outsourcing) oder an andere Produktionsstandorte, erfordern eine Umgestaltung des entsprechenden Geschäftsprozesses. Der Einsatz neuer Maschinen setzt im Allgemeinen ebenfalls Prozessanpassungen voraus. Ebenso verhält es sich bei der Weiterentwicklung oder der Überarbeitung von Produkten (z. B. nach einer Rückrufaktion). Die Einführung neuer Gesetze oder Gesetzesänderungen können die Anpassung eines Geschäftsprozesses erforderlich machen, damit die neuen Vorschriften berücksichtigt werden können. In einer SOA-Umgebung macht das Einbeziehen weiterer Services oder der Austausch einzelner Dienste eine Modifikation des Prozesses notwendig.

Damit auch das PMS nach dem neuen Ablauf handeln kann, ist die Fähigkeit eines PMS, das Prozessschema im Rahmen einer Schemaevolution modifizieren zu können, ein Muss [CCPP98, AM00]. Alle neu gestarteten Prozessinstanzen werden dann nach der neuen Version ablaufen. Die spannende Frage ist, was mit den Instanzen geschieht, die noch auf der alten Version basieren. Für kurz laufende Prozesse ist es eventuell möglich, die bereits gestarteten Prozesse auf dem alten Schema zu Ende laufen zu lassen. Im Falle einer Schemaänderung aus Gründen einer Prozesskorrektur, oder allgemein für lang laufende Prozesse, wie z. B. Leasing-Verträge, ist es unumgänglich, dass die Schemaanpassungen auch auf den bereits aktiven Instanzen nachgezogen werden. Das Anpassen des Laufzeitschemas einer Instanz an die neue Version ihrer Vorlage wird Migration genannt [KG99, SO99]. Allerdings kann nicht jede aktive Instanz migriert werden. Voraussetzung dafür ist, dass eine Instanz nicht zu weit in ihrer Ausführung fortgeschritten ist. Sie muss verträglich (engl.: *compliant*) mit den Änderungen sein (vgl. [RRD02]). Unverträgliche Instanzen zu migrieren kann zu Konflikten im weiteren Prozessverlauf führen, wie z. B. nicht

[3] Die Bearbeiterzuordnung und weitere in der Prozessvorlage hinterlegte Vorgaben, wie z. B. die Zeitabhängigkeiten, wurden in der Abbildung nicht explizit dargestellt, da sie für diese Diplomarbeit im weiteren nicht relevant sind.

korrekt versorgte Daten oder von der (neuen) Vorlage abweichende Ausführungsreihenfolgen der Arbeitsschritte.

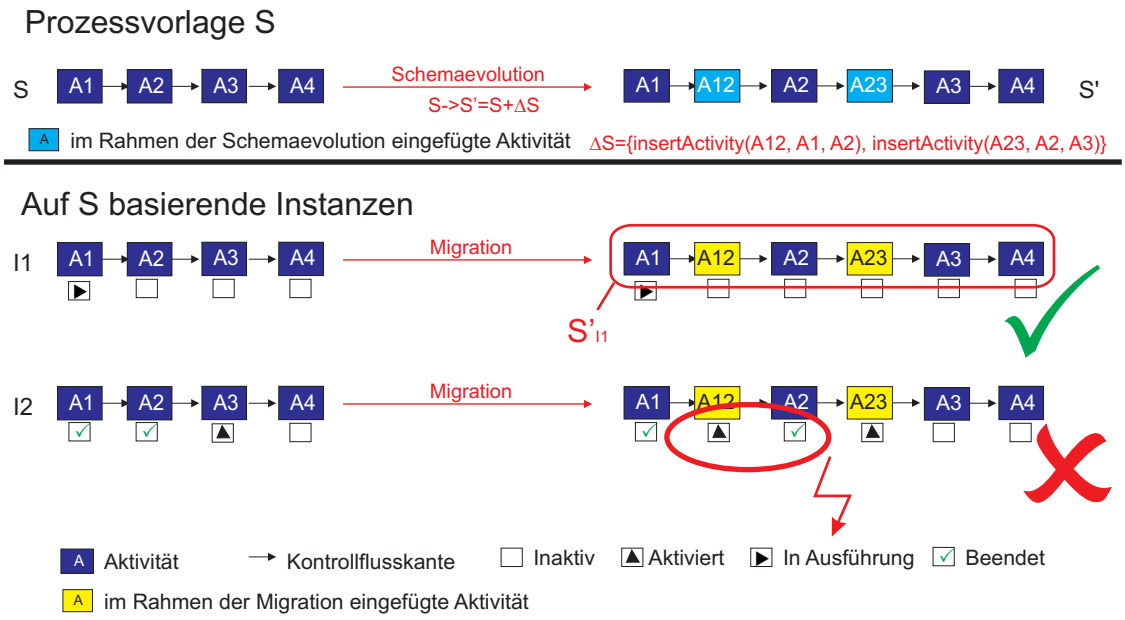

Abbildung 1.3: Schemaevolution von S nach S' mit anschließender Migration der Instanzen

Die Instanz I1 aus Abbildung 1.3 kann auf die im Rahmen der Schemaevolution aus dem Schema S neu hervorgegangene Schemaversion S' migriert werden, indem die Schemaänderung ΔS – logisch gesehen – ebenfalls auf das Laufzeitschema S_{I1} der Instanz I1 angewendet wird. Die neue Version S'_{I1} des Laufzeitschemas der Instanz I1 stimmt daraufhin mit der neuen Version S' der Vorlage überein. Wird dieselbe Änderung jedoch auf das Laufzeitschema S_{I2} von I2 angewendet, so kommt es zu einem Zustandskonflikt. Die Aktivität A2 ist dann bereits beendet worden, obwohl die Vorgängeraktivität A12 noch nicht einmal gestartet worden ist. I2 ist somit mit der Änderung ΔS nicht verträglich und darf nicht migriert werden.

Abbildung 1.4 demonstriert eine weitere, von einem PMS unbedingt zu unterstützende Änderungsart: Das Laufzeitschema S_{I1} der Instanz I1 wird individuell und dynamisch zur Laufzeit gegenüber der Vorlage S abgeändert, indem die Aktivität A34 zwischen A3 und A4 eingefügt wird. Der Prozessablauf dieser Instanz weicht somit von dem durch die Prozessvorlage vorgegebenen Ablauf ab: Die Instanz ist mit dem *Bias* $\Delta S_{I1} = \{insertActivity(A34,\ A3,\ A4)\}$ gegenüber ihrer Vorlage S verzerrt. Der Bias beschreibt die Menge der instanz-spezifischen Änderungsoperationen, die auf das Laufzeitschema der Instanz angewendet worden sind [RRD03b]. Das resultierende Laufzeitschema $S_{I1}*$ stimmt nicht mehr mit der Vorlage S überein, die Instanz basiert aber weiterhin darauf. Die anderen Instanzen gleichen Typs – wie z. B. Instanz I2 aus Abbildung 1.4 – sind davon nicht betroffen.

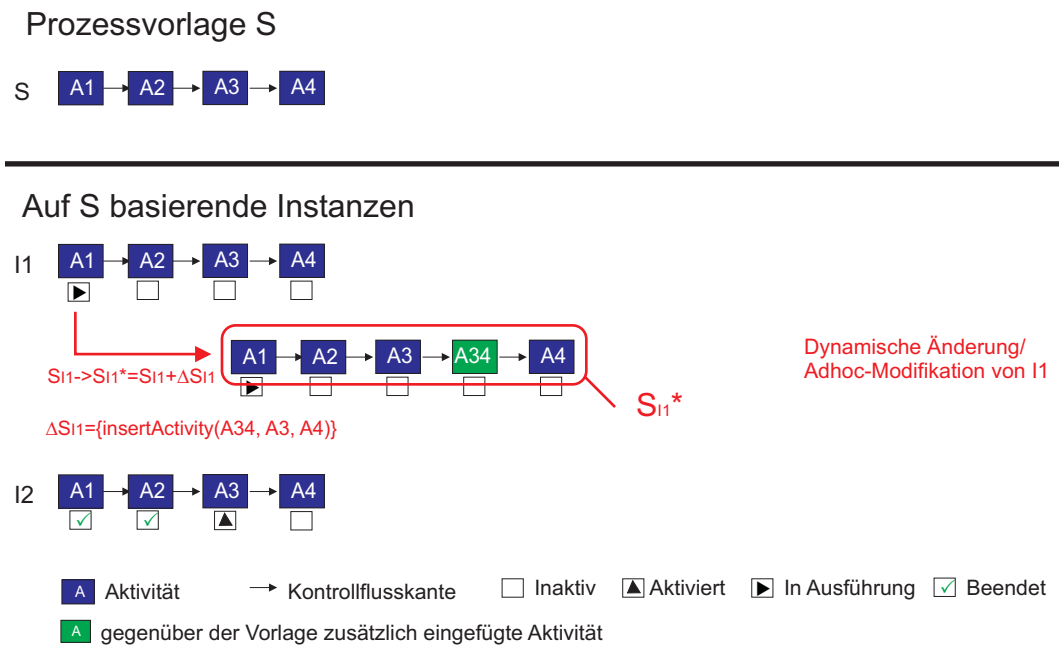

Abbildung 1.4: Dynamische Änderung/Ad-hoc-Modifikation einer Instanz

Mit dynamischen Änderungen einer Instanz, auch Ad-hoc-Modifikationen genannt, können individuelle Prozessanpassungen durchgeführt werden [RD98]. Individuelle Prozessanpassungen sind u. a. notwendig, wenn vorliegende Umstände oder die gegebene Situation es nicht erlauben, wie geplant, d. h. nach Vorlage, vorzugehen. Zum Beispiel muss in einem Krankenhaus oft von dem generell festgelegten Untersuchungs- oder Operationsablauf abgewichen werden, weil weitere Leiden des Patienten die normale Vorgehensweise nicht zulassen. Die zusätzlichen Maßnahmen sind von Patient zu Patient unterschiedlich, d. h. für jeden Patienten muss der Prozess individuell angepasst werden. Dies erfolgt durch die dynamischen Änderungen der den Patienten zugeordneten Prozessinstanzen.

In der Produktion kann der Ausfall einer Maschine eine Ad-hoc-Modifikation nach sich ziehen, wenn nicht schon für diesen Notfall in der Prozessvorlage ein Notplan hinterlegt ist. Oft können nicht für alle Ausnahmesituationen oder auftretende Ereignisse in der Prozessvorlage entsprechende Reaktionsschritte[4] vorgesehen werden, weil entweder zu viele Ausnahmen bzw. Ereignisse zu berücksichtigen wären oder weil einfach nicht alle in der Praxis auftretbaren Konflikte und Ereignisse vorhersagbar sind [RD98, Aa01].

Ein adaptives PMS muss also beide Arten von Änderungen – Ad-hoc-Änderungen einer Instanz und Prozesstyp-Änderung – zulassen, um wirklich praxistauglich zu sein. Darüber hinaus muss

[4] Arbeitsschritte, um der Ausnahmesituation zu begegnen oder um auf eingetretene Ereignisse zu reagieren

das PMS auch das Zusammenspiel beider Änderungsarten unterstützen, d. h. es muss möglich sein, die Prozesstypänderungen auch auf bereits ad hoc modifizierte Instanzen zu propagieren.

Begünstigt wird die Adaptivität bei den Prozess-Management-Systemen dadurch, dass die relevanten Prozesse nicht implizit im Anwendungscode versteckt sind, wie es bei properitären Applikationen der Fall ist, sondern explizit mit einer Beschreibungssprache, meist auf Graphen basierend, im System hinterlegt werden [RD02, S. 1-51f., S. 1-60]. So sind nicht nur die Auswirkungungen einer Änderung besser zu erfassen, es muss bei einer Prozessänderung auch nicht der Programmcode des Systems umgeschrieben werden, was umfassende Tests nach sich ziehen würde. Stattdessen muss lediglich die Repräsentation des Prozesses angepasst werden. Gehorcht die Repräsentionsform bestimmten Design- und Korrektheitsregeln, so kann damit schon bestimmt werden – und das automatisch vom System –, ob die Änderungen überhaupt erlaubt sind.

Doch obwohl die Prozess-Management-Systeme diesen Vorteil gegenüber den properitären Anwendungen aufweisen und damit Adaptiviät begünstigen, wird bei den kommerziell erhältlichen Systemen die Adaptivität nur rudimentär unterstützt (vgl. z. B. [AM00], [RD02, S. 7-2]).

1.3 Adaptivität und kommerzielle PMS

Die meisten kommerziellen Prozess-Management-Systeme unterstützen das Abändern einer Prozessvorlage als eine Form der Adaptivität. Die Prozessvorlagen können abgeändert werden, zukünftige Prozessinstanzen halten sich an die neuen Ablaufvorgaben. Viele dieser PMS, wie z. B. MQSeries Workflow [Re00], lassen aber die bereits aktiven Instanzen bei einer Schemaänderung unberücksichtigt, d. h. führen sie weiterhin nach dem Prozessschema der alten Vorlage aus. Prozesskorrekturen, Optimierungen oder anderweitige wichtige Anpassungen werden bei diesen Instanzen nicht berücksichtigt.

Andere PMS bieten dagegen an, die bereits aktiven Prozesse zurückzusetzen, um sie erneut, aber diesmal nach der neuen Version der Vorlage auszuführen. In vielen Fällen ist jedoch das Zurücksetzten überhaupt nicht möglich – eingesetzte Programme bieten z. B. keine Undo-Funktionen bzw. Kompensationsfunktionen – oder es ist nicht praktikabel.

Wieder andere PMS-Vertreter übernehmen zwar die Änderungen auf die aktiven Instanzen, führen aber keine Konsistenzprüfungen durch, wie z. B. Staffware [Re00]. Sie lassen auch Änderungen zu, die mit dem bereits erreichten Ausführungsstand nicht verträglich sind, wie z. B. das Einfügen von Aktivitäten in bereits ausgeführte Prozessabschnitte. Probleme im weiteren Verlauf bis hin zu Programmabstürzen können die Folge sein.

Bei der Unterstützung der dynamischen Änderung einzelner Instanzen liegt auf dem Markt eine ähnliche Situation vor wie bei der Schemaevolution. Die Prozess-Management-Systeme lassen entweder Modifikationen überhaupt nicht zu oder aber auch solche, die zu strukturellen und/oder zustandsbedingten Konflikten bei den resultierenden Laufzeitschemata der Instanzen führen [Re00]. Die Migration von gegenüber der Vorlage verzerrten Instanzen ist zur Zeit weder in

kommerziellen Produkten realisiert, noch existieren bisher theoretische Ansätze, die dieses Ziel verfolgen.

Die Abteilung Datenbanken- und Informationssysteme der Universität Ulm erarbeitet im Rahmen eines DFG-Forschungsprojekts und des Verbundprojekts „AristaFlow“ zusammen mit anderen Universitäten und mit Partnern aus der Industrie die Konzepte eines voll adaptiven Prozess-Management-Systems.

Dabei lauten die zentralen Fragestellungen: Wie ist das konzeptionelle Vorgehen bei einer Schemaevolution, Migration und einer dynamischen Änderung einer Instanz? Welche Korrektheitskriterien sind dabei einzuhaltenden? Wie sehen die Algorithmen aus, welche die Einhaltung der Kriterien überprüfen? Die Fragestellungen werden in diversen Veröffentlichungen der Abteilung ausführlich behandelt. Kapitel 2 geht auf die zugrundeliegenden Problemstellungen näher ein.

Die erarbeiteten Konzepte werden in dem Forschungssystem ADEPT 2 umgesetzt[5].

1.4 Problemstellung und Fokus der Arbeit

Bei der Realisierung komplexer, parallel verarbeitender Systeme, wie Prozess-Management-Systeme und Datenbanken, treten neben den konzeptionellen Fragen auch wichtige Fragen bezüglich einer praxistauglichen Implementierung auf: Wie können die theoretischen Konzepte effizient in die Praxis umgesetzt werden? Wie kann mit den knappen Ressourcen heutiger Rechner, wie z. B. dem Arbeitsspeicher, umgegangen werden? Welche Probleme bringt die Parallelverarbeitung mit sich? Wie können diese verhindert werden?

Ziel dieser Diplomarbeit ist es, diese Fragen für die Umsetzung der Adaptivität in Prozess-Management-Systemen zu beantworten, um eine effiziente Realisierung der Prozess-Schemaevolution in Hochleistungs-Prozess-Management-Systemen zu erreichen. Die Arbeit wird verschiedene Lösungsansätze herleiten und auf die Praxistauglichkeit hin diskutieren. Insbesondere wird darauf eingegangen,

- wie Prozessvorlagen und -instanzen ressourcensparend intern repräsentiert werden können
- wie die Schemaevolution, Migration und die dynamische Änderung einer Instanz physisch realisiert werden
- wie bei der gegebenen Architektur die Migration effizient durchgeführt werden kann
- in welchem Umfang Maßnahmen zur Steuerung von konkurrierenden Zugriffen auf die Datenstrukturen zu treffen sind
- wie diese Maßnahmen konkret aussehen

[5] ADEPT: Application Development Based on Encapsulated Pre-Modeled Process Templates

Auch wenn die Preise für Arbeitsspeicher in den letzten Jahren dramatisch gesunken sind, weist diese Speichergattung trotzdem noch verhältnismäßig hohe Anschaffungskosten auf. Deshalb ist er auch noch in der heutigen Zeit eine verhältnismäßig knappe Ressource, mit der sparsam umgegangen werden muss. In realen Anwendungsszenarien ist eine Anzahl an aktiven Instanzen im Tausenderbereich keine Seltenheit. Bei dieser Größenordnung ist es deshalb unumgänglich, dass die Datenstrukturen für die Repräsentation der Vorlagen und Instanzen möglichst geringen Speicherbedarf aufweisen, um die Restriktionen bezüglich des Speicherbedarfs zu erfüllen. Die große Zahl an Instanzen setzt auch eine effiziente Umsetzung der Migrationsstrategie voraus, damit die parallel ablaufenden Funktionen des Prozess-Management-Systems, wie das Zeitmanagement, die Benutzeranmeldung und -abmeldung, das Weiterschalten von Instanzen, etc., während eines Migrationslaufes nicht beeinträchtigt werden. An einem Prozess-Management-System sind viele Benutzer gleichzeitig aktiv und es finden viele Aktionen parallel statt. So kann es auch vorkommen, dass mehrere Benutzer versuchen, dasselbe Schema oder die gleiche Instanz parallel abzuändern. Oder ein Benutzer will eine Instanz abändern, die momentan in Begriff ist zu migrieren. Zu untersuchen ist, ob und inwieweit diese konkurrierenden Zugriffe Probleme verursachen und wie in sinnvoller Weise Maßnahmen dagegen getroffen werden können.

Der Aufbau der Diplomarbeit richtet sich nach den oben angesprochenen Fragen, die für eine Implementierung beantwortet werden müssen. Kapitel 3 leitet einen Ansatz zur Repräsentation von Prozessvorlagen und -instanzen her, der die Forderung an geringen Speicherbedarf erfüllt und bei dem die Schemaevolution, die Migration und eine dynamische Änderung einer Instanz auf effiziente Weise umgesetzt wird. Im darauf folgenden Kapitel 4 werden zwei Implementierungsvarianten vorgestellt und auf Tauglichkeit diskutiert, die das Ziel haben, die Migration zu beschleunigen. Kapitel 5 beschäftigt sich mit der Fragestellung der konkurrierenden Zugriffe.

Zuvor wird jedoch in Kapitel 2 vertiefend auf die Theorie hinter der Schemaevolution, der Migration und den dynamischen Änderungen einer Instanz eingegangen, um die Grundlagen für die folgenden Kapitel zu legen.

Kapitel 6 reflektiert die gewonnenen Erkenntnisse und beleuchtet diese im Licht der an die Diplomarbeit gestellten Ziele.

Kapitel 2

Konzepte des adaptiven Prozess-Managements

Dieses Kapitel gibt eine kurze Einführung in die Konzepte des adaptiven Prozess-Managements. Es wird vertiefend auf die Schemaevolution, auf die Migration und auf die dynamischen Änderungen einer Instanz eingegangen und aufgezeigt, welche Konflikte dabei auftreten können und welche sich aus dem Zusammenspiel dieser Mechanismen ergeben können. Darauf aufbauend wird hergeleitet, welche Tests notwendig sind, um eine korrekte Schemaevolution, Migration und Instanzänderung zu garantieren und wie schließlich ein konsistenter Ablauf einer Schemaevolution, einer Migration und einer dynamischen Änderung einer Instanz aussehen kann.

2.1 Schemaevolution

Bei der Schemaevolution [CCPP98, KG99] wird, wie in Kapitel 1 schon angesprochen, das typbeschreibende Prozessschema S innerhalb einer Änderungstransaktion durch Anwendung von kontrollfluss-ändernden, datenfluss-ändernden und/oder attribut-ändernden Operationen modifiziert und dadurch in eine neue Version $S' = S+\Delta S$ überführt. ΔS repräsentiert die Abweichung von S' gegenüber S, d. h. ΔS beschreibt die Wirkung der Änderungsoperationen auf das Schema S.

Die Kapselung der Änderungsoperationen in einer Transaktion garantiert, dass bei einem Commit durch den Benutzer – und nur dann – das durch diese Operationen entstandene neue Schema vollständig und dauerhaft ins Prozess-Management-System übernommen wird.

Jedoch überführen nicht jede beliebige Modifikationen ein korrektes Prozessschema in ein neues, ebenfalls korrektes Schema.

2.1.1 Korrekte Schemata

Ein korrektes Schema liegt genau dann vor, wenn keine Korrektheitskriterien des zugrundeliegenden Prozess-Modells für Schemata verletzt werden. In ADEPT gelten als Korrektheitskriterien die in [Re00, Kapitel 3 ff.] aufgestellten Strukturierungsregeln für den Kontroll- und Datenfluss. Zwei Regeln, die erfüllt sein müssen, sollen exemplarisch herausgegriffen werden:

1. Der dem Schema zugrundeliegende Prozessgraph muss zyklenfrei bezüglich Kontroll- und Sync-Kanten sein ([Re00, Strukturierungsregel $KF-4^*$]).
2. Für jede Aktivität, die obligat lesend auf ein Datenelement zugreift, muss auf jedem möglichen Ausführungspfad eine Vorgängeraktivität existieren, die das entsprechende Datenelement obligat schreibt ([Re00, Strukturierungsregel DF-1]).

Inkorrekte Schemata dürfen nicht in das Prozess-Management-System übernommen werden, da es bei ihnen zu Problemen bei der Ausführung kommen kann: Ein Verstoß gegen die geforderte Zyklenfreiheit des Kontrollflusses führt bei der Prozessausführung zu Verklemmungen (engl.: *Deadlocks*) und verhindert somit die korrekte Beendigung des Prozesses[1]. Ist die zweite genannte Regel nicht erfüllt, so kann in dem Datenelement zum Zeitpunkt des Lesens ein undefinierter Wert stehen. Die korrekte Ausführung eines mit der Leseraktivität assoziierten Programms kann dann aufgrund des undefinierten Wertes nicht garantiert werden. Falsche Ergebnisse oder Programmabstürze können die Folge sein.

Unter anderem kann ein unkontrolliertes Einfügen von Sync-Kanten im Rahmen einer Schemaevolution zu inkorrekten Versionen von Prozessvorlagen führen, wie anhand dem Beispiel aus Abbildung 2.1 exemplarisch gezeigt werden kann:

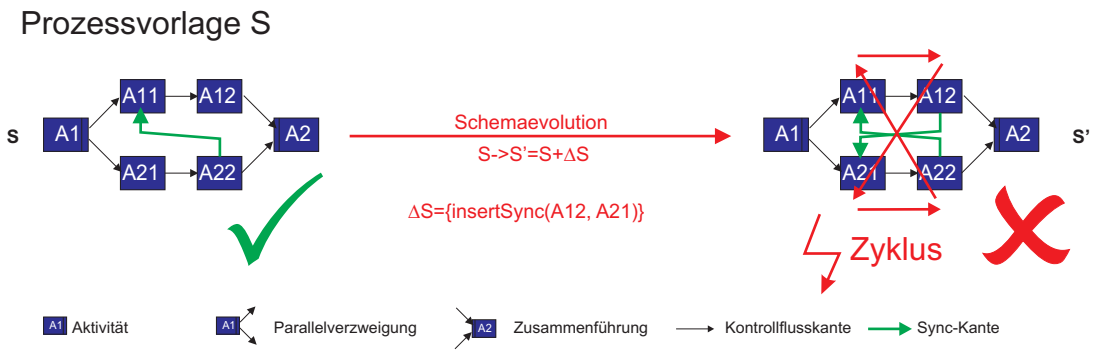

Abbildung 2.1: Zyklus im Prozessgraph aufgrund unkontrollierten Einfügens einer Sync-Kante

Die neue Version S' geht durch das Einfügen der Sync-Kante A12→A21 aus der vorigen Version S hervor. Aufgrund der neuen Sync-Kante enthält die Prozessvorlage nun den Zyklus

[1] Ein Prozess gilt als korrekt beendet, wenn ein definierter Endzustand eingenommen wurde.

A11→A12→A21→A22→A11. Damit verstößt das Schema S' gegen die oben genannte Regel, dass der dem Schema zugrunde liegende Prozessgraph keine Zyklen bezüglich Kontroll- und Sync-Kanten aufweisen darf. S' stellt somit kein korrektes Schema dar. Probleme bei der Prozessausführung sind die Folge. Der Zyklus führt bei der Ausführung einer darauf basierenden

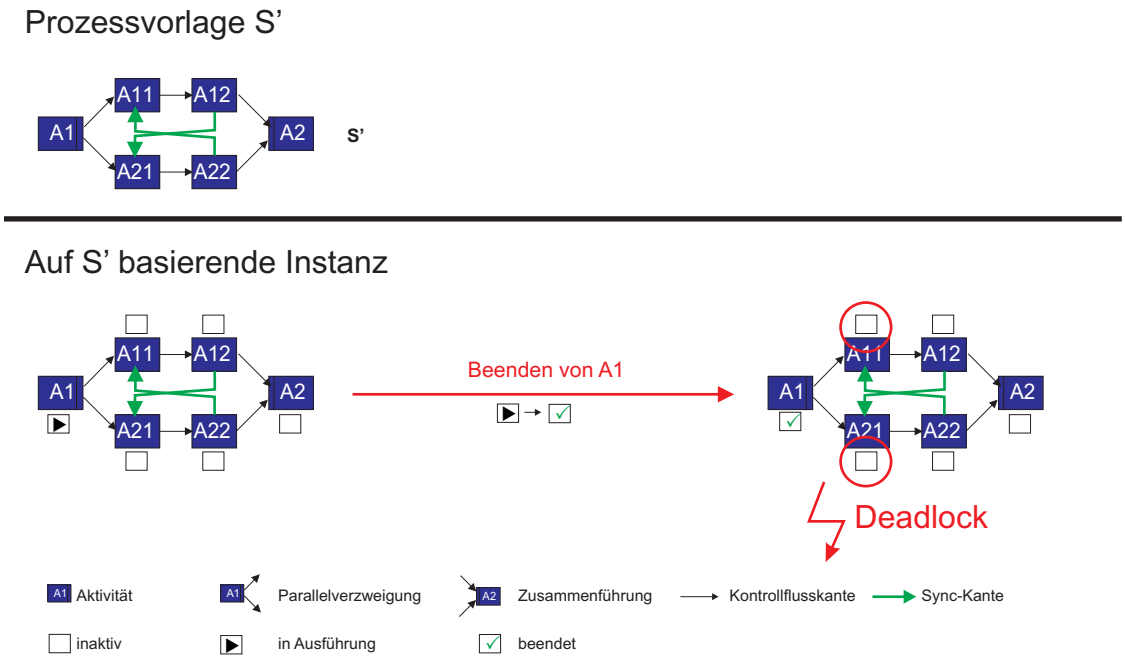

Abbildung 2.2: Deadlock bei der Prozessausführung aufgrund eines Zyklus im Prozessgraph

Prozessinstanz zu einem Deadlock: Bei der auf S' basierenden Instanz aus Abbildung 2.2 kann A11 nach erfolgter Abarbeitung der Aktivität A1 nicht aktiviert werden, da sie aufgrund der Sync-Kante A22→A11 auf die erfolgreiche Ausführung von A22 wartet. A22 kann aber erst nach A21 ausgeführt werden. Somit wartet A11 auf die erfolgte Ausführung von A21. A21 kann jedoch nicht aktiviert werden, da sie selbst aufgrund der anderen Sync-Kante A12→A21 auf die erfolgreiche Beendigung der Aktivität A12 wartet. Da aber A12 erst nach A11 gestartet werden kann, muss A21 transitiv auf A11 warten. A11 und A21 warten also gegenseitig aufeinander. Damit liegt ein Deadlock vor und der Prozess kommt an dieser Stelle zum Erliegen. Der Prozess kann somit nicht korrekt beendet werden.

Das PMS muss im Rahmen der Commit-Behandlung einer Änderungstransaktion in einem Korrektheitstest sicherstellen, dass die Modifikationen nicht zu einem inkorrekten Schema führen. In [Re00] werden effiziente Verfahren vorgestellt, mit denen die Einhaltung der oben genannten Regeln geprüft werden können. Scheitert der Test, so muss die Änderungstransaktion mit einem entsprechenden Hinweis zurückgewiesen werden. Der Benutzer muss die Möglichkeit haben, Korrekturen durchzuführen. Alternativ kann stets vor einer Anwendung einer Änderungsoperation getestet werden, ob es durch die Änderung nicht zu einem inkorrekten Schema kommt, um die

Änderungsoperation bei Bedarf sogleich zurückzuweisen. Der Unterschied zwischen den beiden Methoden ist, dass bei der ersten Methode erst nach der Anwendung aller Operationen einer Änderungstransaktion ein korrektes Schema vorliegen muss, während bei der zweiten nach jeder einzelnen Operation die Bedingungen an das bis dahin entstandene Schema erfüllt sein müssen.

2.1.2 Ablauf einer Schemaevolution

Wird der Korrektheitstest im Rahmen der Commit-Behandlung durchgeführt, so ergibt sich für die Schemaevolution ein Ablauf, der sich mit dem Zustandsdiagramm aus Abbildung 2.3 beschreiben lässt: Mit dem Öffnen einer Änderungstransaktion durch den Benutzer tritt ein

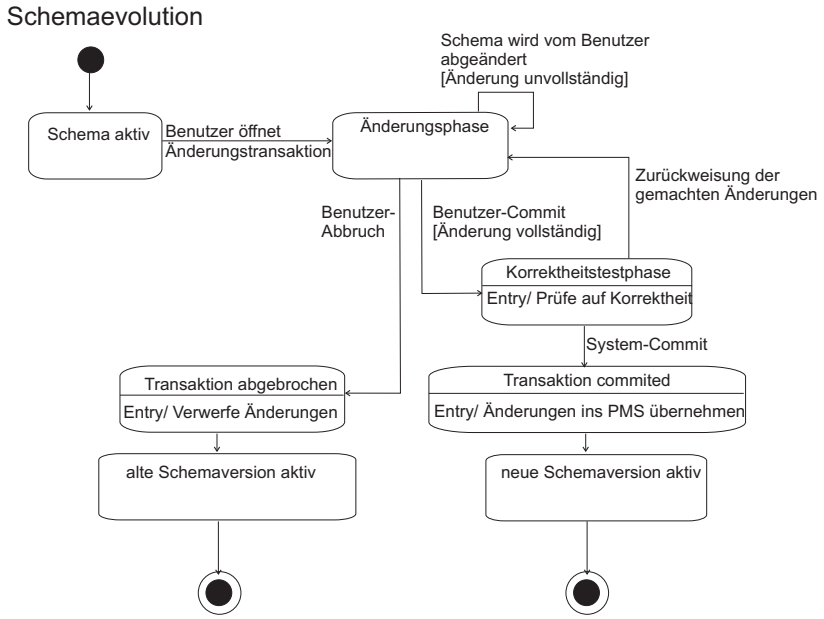

Abbildung 2.3: Zustandsdiagramm für eine Schemaevolution

aktives Schema in die Änderungsphase ein. In dieser Phase führt der Benutzer die gewünschten Änderungen durch. Bei einem Benutzerabbruch werden sämtliche Modifikationen des Schemas rückgängig gemacht. Führt der Benutzer ein Commit der Anpassungen durch, so tritt die neue Version des Schemas in die Phase ein, in der das Schema auf Verstöße gegen die dem Prozess-Modell zugrunde gelegten Korrektheitskriterien getestet wird. Treten Fehler auf, so wird der Benutzer aufgefordert, Korrekturen am vorliegenden Schema vorzunehmen. Konnte der Test ohne Beanstandungen beendet werden, so commited auch das System die Transaktion, was die dauerhafte Übernahme ins Prozess-Management-System zur Folge hat.

Konnte die neue Version S' des Schemas erfolgreich ins Prozess-Management-System übernommen werden, so laufen alle neu gestarteten Instanzen des entsprechenden Typs fortan danach

ab[2]. Interessant ist die Frage, wie mit den Instanzen umgegangen werden soll, die noch auf Grundlage der alten Version S gestartet wurden.

2.2 Instanzen und Schemaevolution

Eine Möglichkeit, mit Instanzen bei einer Schemaevolution umzugehen, die noch auf der Grundlage der alten Version gestartet worden sind, ist es, die schon aktiven Instanzen von der Schema-Änderung unberücksichtigt zu lassen [SO99]. Für besonders kurzlebige Prozesse mag diese Vorgehensweise noch akzeptabel sein, aber bei langlebigen Instanzen können dadurch Probleme entstehen. Ein Beispiel: Der Zulassungsprozess von neu entwickelten Medikamenten geht über Jahre und muss sich streng an die vom Gesetzgeber vorgeschriebenen Regeln halten. Bedient sich nun das PMS nur der beschriebenen Methode, so könnten neu auferlegte Bestimmungen nicht mehr auf die bereits laufenden Zulassungsverfahren angewendet werden und der Pharma-Konzern würde sich strafbar machen. Bei wenigen betroffenen Instanzen besteht die Möglichkeit, durch Ad-hoc-Modifikationen die Instanzen per Hand auf die neuen Gegebenheiten anzupassen, aber bei tausenden betroffener Instanzen ist dies unrealistisch. Außerdem ist die manuelle Anpassung von mehreren Instanzen fehleranfällig. Eine weitere Möglichkeit, mit bereits laufenden Instanzen nach einer Schemaänderung umzugehen, ist es, diese abzubrechen und bezüglich des neuen Schemas erneut auszuführen [SO99]. Aber das Zulassungsverfahren abzubrechen und dann alle Untersuchungen erneut durchzuführen ist nicht praktikabel.

Um die Effekte der beiden bis jetzt vorgestellten Verfahren abzumildern, wird ein komplexer Prozess oft in mehrere, kurzlebige Abschnitte aufgesplittet, die durch separate “Mini-Schemata” beschrieben werden. Die erste vorgestellte Methode kann dann aufgrund der Kurzlebigkeit eines Abschnittes eher auf den Abschnitt angewendet werden, als auf den ganzheitlichen Prozess. Bei Anwendung der zweiten Methode wird nicht der ganze Prozess zurückgesetzt, sondern nur der jeweilige Abschnitt. Abgesehen davon, dass mehrere Prozessfragmente schwer zu handhaben sind und sie nicht die natürliche, zusammenhängende Sicht auf den Prozess bieten, sind auch Änderungen, die gleichzeitig mehrere Abschnitte betreffen, unmöglich, da jeder Abschnitt seine eigene Prozessvorlage besitzt.

Die für die Praxis interessanteste Vorgehensweise, mit den bereits laufenden Instanzen bei einer Schemaänderung umzugehen, ist die *Migration*.

[2]Eventuell kann noch ein Zeitpunkt angegeben werden, ab dem die neue Version Gültigkeit besitzt. Erst die nach diesem Zeitpunkt gestarteten Instanzen werden dann nach der neuen Version der Vorlage ablaufen. Die anderen, zuvor gestarteten Instanzen werden noch bis zu diesem Zeitpunkt nach der alten Version ablaufen.

2.3 Migration

Bei der Migration wird, wie in Kapitel 1 schon beschrieben, im Anschluss an die Schemaevolution versucht, die Laufzeitschemata der Instanzen, die noch auf der alten Version S der geänderten Vorlage gestartet wurden, an die neue Version S' anzupassen [SO99, KG99].

Bei der Vorgehensweise muss zwischen unverzerrten und verzerrten Instanzen unterschieden werden.

2.3.1 Migration unverzerrter Instanzen

Das Laufzeitschema einer unverzerrten Instanz I stimmt mit dem Schema S der Vorlage überein (formal: $S_I \equiv_{trace} S$)[3]. Deshalb wird eine unverzerrte Instanz auf die neue Version S' angepasst, indem die Schemaänderungen ΔS – logisch gesehen – auf dem Laufzeitschema S_I von I nachgezogen werden. Nach der Migration stimmt dann das Laufzeitschema S'_I dieser Instanz I mit der neuen Version S' der Vorlage überein: $S'_I = S_I + \Delta S \equiv_{trace} S + \Delta S = S'$. Die Instanz ist damit wiederum unverzerrt gegenüber ihrer Vorlage.

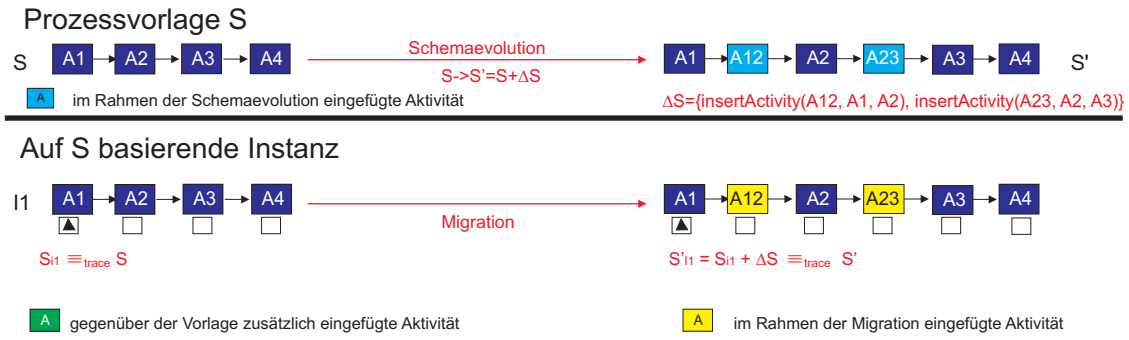

Abbildung 2.4: Beispielmigration einer unverzerrten Instanz

Die Instanz I1 aus Abbildung 2.4 ist z. B. gegenüber dem Schema S der Prozessvorlage nicht verzerrt (es wurden keine zusätzlichen Aktivitäten eingefügt; $\Delta S_{I1} = \emptyset$). Um die Instanz an die neue Version S' des Schemas, die aus S durch das Einfügen der Aktivität A12 zwischen A1 und A2 und der Aktivität A23 zwischen A2 und A3 hervorging ($\Delta S = \{insertActivity(A12, A1, A2), insertActivity(A23, A2, A3)\}$), anzupassen, muss die ganze Schemaänderung ΔS auf I1 nachgezogen werden.

[3]Sind zwei Schemata ausführungsäquivalent (formal: $\equiv_{trace}$), so können alle Ausführungshistorien, die auf dem einen Schema erzeugbar sind, auch auf dem anderen erzeugt werden und umgekehrt.

2.3.2 Migration verzerrter Instanzen

Verzerrte Instanzen sind in ihrem Laufzeitschema S_I* gegenüber ihrer Vorlage S abgeändert, d. h. $S_I* = S_I + \Delta S_I \equiv_{trace} S + \Delta S_I$. Die Schemaänderungen ΔS können sich dann mit den Instanzänderungen ΔS_I überlappen (formal: $\Delta S \cap \Delta S_I \neq \emptyset$), d. h. die Schemaänderungen ΔS bewirken auf das Laufzeitschema einer unverzerrten Instanz desselben Prozesstyps (teilweise) dieselben Effekte, wie die Instanzänderungen ΔS_I auf das ursprüngliche Laufzeitschema $S_I \equiv_{trace} S$ der Instanz I gehabt haben. Bei verzerrten Instanzen hängt deshalb die Vorgehensweise zur Anpassung an die neue Version von den Beziehungen ab, in denen die Auswirkungen der Instanzänderungen und der Schemaänderungen stehen.

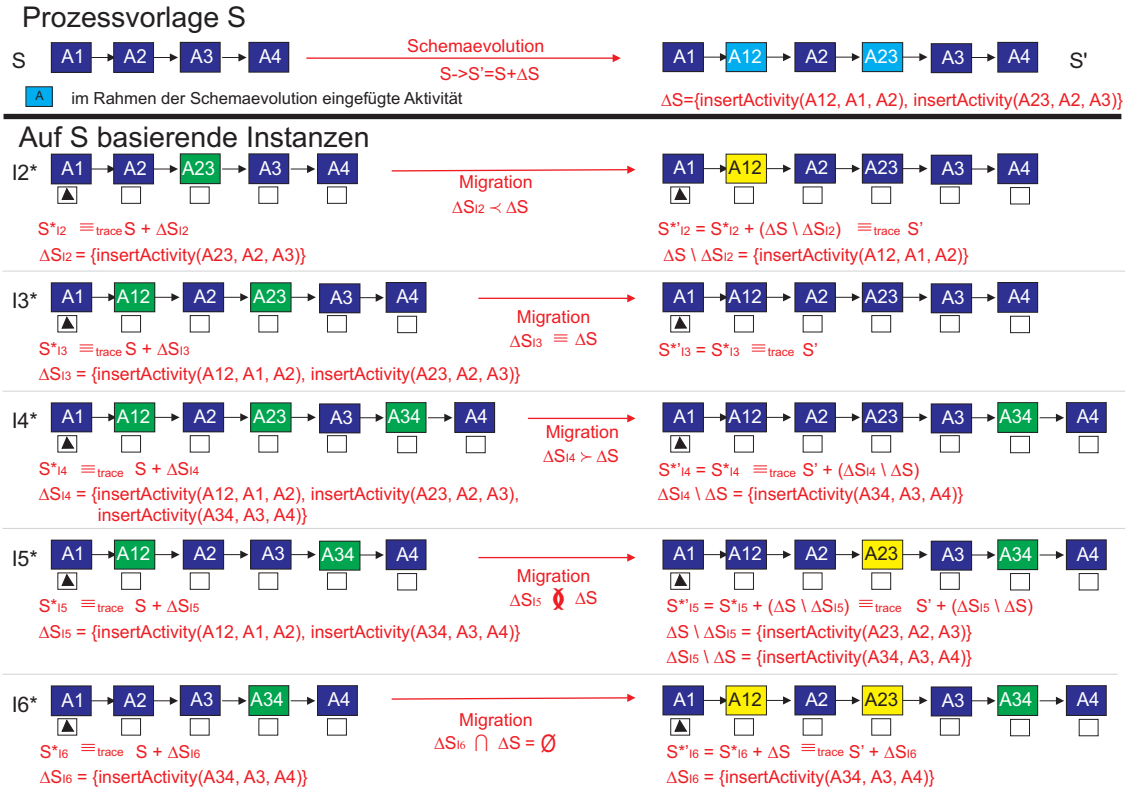

Abbildung 2.5: Beispielmigration verzerrter Instanzen mit unterschiedlichem Überlappungsgrad

Sind die Auswirkungen ΔS_I der Instanzänderungen, mit denen die Instanz I gegenüber S abgeändert wurde, und die Auswirkungen ΔS der Schemaänderungen äquivalent, d. h. $\Delta S_I \equiv \Delta S$, so ist das Laufzeitschema S_I* der verzerrten Instanz I nach [RRD03a] ausführungsäquivalent (englisch: trace equivalent/execution equivalent) zu dem Schema S' der neuen Vorlagenversion: Auf S_I* können dieselben Ablaufhistorien erzeugt werden wie auf S' und umgekehrt [RRD03a]. Damit ist das Einspielen der Änderungsoperationen auf S_I*, die S nach S' geändert haben, nicht mehr notwendig, um die Instanz auf die neue Vorlage anzupassen. Die Instanz I ist bereits

mit der neuen Vorlage konform. Mehr noch: I ist auch nicht mehr verzerrt gegenüber der neuen Vorlage: $S_I{*}' = S_I{*} = S_I' \equiv_{trace} S'$.

Bei der Instanz I3 aus Abbildung 2.5 ist die zu der Schemaänderung ΔS äquivalente Ad-hoc-Modifikation $\Delta S_{I3} = \{insertActivity(A12,\ A1,\ A2), insertActivity(A23,\ A2,\ A3)\}$ vorgenommen worden. Bei der Migration muss somit keine Anpassung von S_{I3} an S' erfolgen. Gegenüber S' ist dann I3 auch nicht mehr verzerrt.

In diesem Beispiel wurde implizit angenommen, dass z. B. die auf Instanzebene eingefügte Aktivität A12 die gleiche ist, wie die auf Typebene eingefügte. In der Praxis können jedoch zwei Aktivitäten, die den gleichen Namen bekommen haben, für unterschiedliche Aktionen stehen. Oder anders herum, es können zwei dem Namen nach unterschiedliche Aktivitäten dennoch dieselben Funktionen erfüllen. Um den Test auf Äquivalenz korrekt durchführen zu können, ist es aber von entscheidender Bedeutung, dies berücksichtigen zu können. Für die Praxis müssen deshalb Kriterien gefunden werden, mit denen man Aktivitäten auf Gleichheit vergleichen kann. Daran muss noch geforscht werden. Aufgrund der Komplexität dieses Problems wird in dieser Diplomarbeit darauf nicht weiter eingegangen und stets zwei im Namen übereinstimmende Aktivitäten als gleich betrachtet.

Neben der Äquivalenz gibt es weitere Beziehungen, in denen ΔS_I und ΔS zueinander stehen können. Je nach Überlappungsgrad muss eine bestimmte Migrationsstrategie angewendet werden [Ri04].

Sind die Änderungen disjunkt, d. h. $\Delta S_I \cap \Delta S = \emptyset$, so bewirken die beiden Änderungsarten ganz unterschiedliche Effekte auf unterschiedliche Bereiche des Laufzeitschemas der Instanz. In diesem Fall muss ganz ΔS analog den unverzerrten Instanzen für die Anpassung eingespielt werden. Instanz I6 aus Abbildung 2.5 zeigt diesen Sachverhalt.

Ist ΔS_I subsumptions-äquivalent zu ΔS, d. h. $\Delta S_I \prec \Delta S$, so schließen die Schemaänderungen die Instanzänderungen komplett ein, aber nehmen noch zusätzliche Modifikationen vor. Logisch gesehen müssen dann die zusätzlichen Modifikationen $\Delta S \setminus \Delta S_I$ bei der Migration noch auf der verzerrten Instanz nachgeholt werden[4]. Nach der Migration stimmt das Laufzeitschema der Instanz mit der neuen Version des Schemas der Vorlage überein. Die Instanz ist nicht mehr verzerrt gegenüber S': $S_I{*}' = S_I' \equiv_{trace} S'$.

Bei Instanz I2 aus Abbildung 2.5 wurde das Einfügen von Aktivität A23 bereits durch die Ad-hoc-Modifikation vorweggenommen. Für die Migration muss dann nur noch A12 zwischen A1 und A2 eingefügt werden. Da auf Instanzebene keine weiteren Änderungen vorgenommen wurden, gilt: $\Delta S_{I2} \prec \Delta S$.

Gilt umgekehrt $\Delta S_I \succ \Delta S$, so sind die Instanzänderungen umfassender als die Schemaänderungen. Für das Laufzeitschema $S_I{*}$ der verzerrten Instanz gilt: $S_I{*} = S_I + \Delta S_I \equiv_{trace} S + \Delta S_I =$

[4] $\Delta S \setminus \Delta S_I$ beschreibt alle Auswirkungen der Schemaänderung ΔS, die nicht auch durch die Instanzänderungen ΔS_I bewirkt werden.

$S + (\Delta S \cup (\Delta S_I \setminus \Delta S)) = S' + (\Delta S_I \setminus \Delta S)$[5]. Damit ist die Instanz bis auf die zusätzlichen Modifikationen $\Delta S_I \setminus \Delta S$ konform zur neuen Version. Für die Migration sind somit auf logischer Ebene keine weiteren Aktionen notwendig und es gilt für das Laufzeitschema S_I*' nach der Migration: $S_I*' = S_I* \equiv_{trace} S' + (\Delta S_I \setminus \Delta S)$.

Im Rahmen der Ad-hoc-Modifikation wurde bei Instanz I4 aus Abbildung 2.5 neben A34 auch die beiden Aktivitäten A12 und A23 eingefügt und zwar an die gleichen Stellen, wie im Rahmen der Schemaevolution auf der Prozessvorlage. Die Schemaänderungen sind somit subsumptionsäquivalent zu den Instanzänderungen. Damit ist I4 bereits konform zu der neuen Schemaversion S', für die Migration ist keine weitere Aktion notwendig. Allerdings ist I4 auch gegenüber S' wegen der zusätzlichen Aktivität A34 verzerrt.

Haben die beiden Änderungsarten ΔS_I und ΔS teilweise dieselben Effekte auf das Ursprungsschema gemeinsam, bewirken darüberhinaus aber auch noch weitere, dazu und zueinander disjunkte Modifikationen, so gelten sie als partiell äquivalent. Die Instanz wird dann migriert, indem logisch die Änderungen $\Delta S \setminus \Delta S_I$ auf der Instanz nachgezogen werden. Nach der Migration ist die Instanz mit dem Bias $(\Delta S_I \setminus \Delta S)$ gegenüber der neuen Vorlage verzerrt.

Die Schemaänderung ΔS und die Instanzänderung ΔS_{I5} der Instanz I5 aus Abbildung 2.5 bewirken beide das Einfügen der Aktivität A12 zwischen A1 und A2, weisen aber beide noch die zusätzlichen Einfüge-Modifikationen von A23 bzw. A34 auf. Damit sind die beiden Änderungsarten partiell äquivalent zueinander. Für die Migration von I5 auf das neue Schema S' muss dann nur noch die fehlende Aktivität A23 eingefügt werden. Die bei der Ad-hoc-Modifikation gegenüber der Schemaänderung zusätzlich eingefügte Aktivität A34 ist dafür verantwortlich, dass I5 auch nach der Migration verzerrt bleibt.

Das ist aber nur ein kleiner Aspekt der partiellen Äquivalenz von Instanz- und Schemaänderungen, wie im folgenden deutlich wird.

Zum Beispiel kann auf Schemaebene eine andere Aktivität an die gleiche Stelle des Prozessgraphen eingefügt werden wie auf der Instanzebene. Da in diesem Fall zwar verschiedene Aktivitäten eingefügt werden, dabei aber jeweils die gleiche Stelle des Prozessgraphen betroffen ist, werden die entsprechenden Instanz- und Schemaänderungen auch als partiell äquivalent betrachtet. Die Migration der entsprechenden Instanz ist dann aber nicht mehr so einfach, wie bei dem obigen Fall der partiellen Äquivalenz geschildert, da Fragen aufkommen, die sich nur mithilfe der Benutzer oder durch für solche Fälle im System hinterlegte Regeln beantworten lassen (vgl. Abbildung 2.6): Sollen im resultierenden Laufzeitschema die beiden Aktivitäten sequentiell hintereinander oder parallel angeordnet werden? Wenn sequentiell, welche soll dann zuerst zur Ausführung kommen? Oder soll keine oder nur eine der beiden Aktivitäten an dieser Stelle ausgeführt werden? Wenn nur eine ausgeführt werden soll, dann stellt sich die Frage: Welche?

Ähnliche Fragen ergeben sich, wenn auf Schemaebene dieselben Aktivitäten an anderen Stellen eingefügt werden als auf Instanzebene. Weitere Fälle, die Fragen aufwerfen, liegen vor, wenn

[5] $\Delta S1 \cup \Delta S2$ beschreibt die Auswirkungen der Änderungen $\Delta S1$ kombiniert mit den Auswirkungen der Änderungen $\Delta S2$.

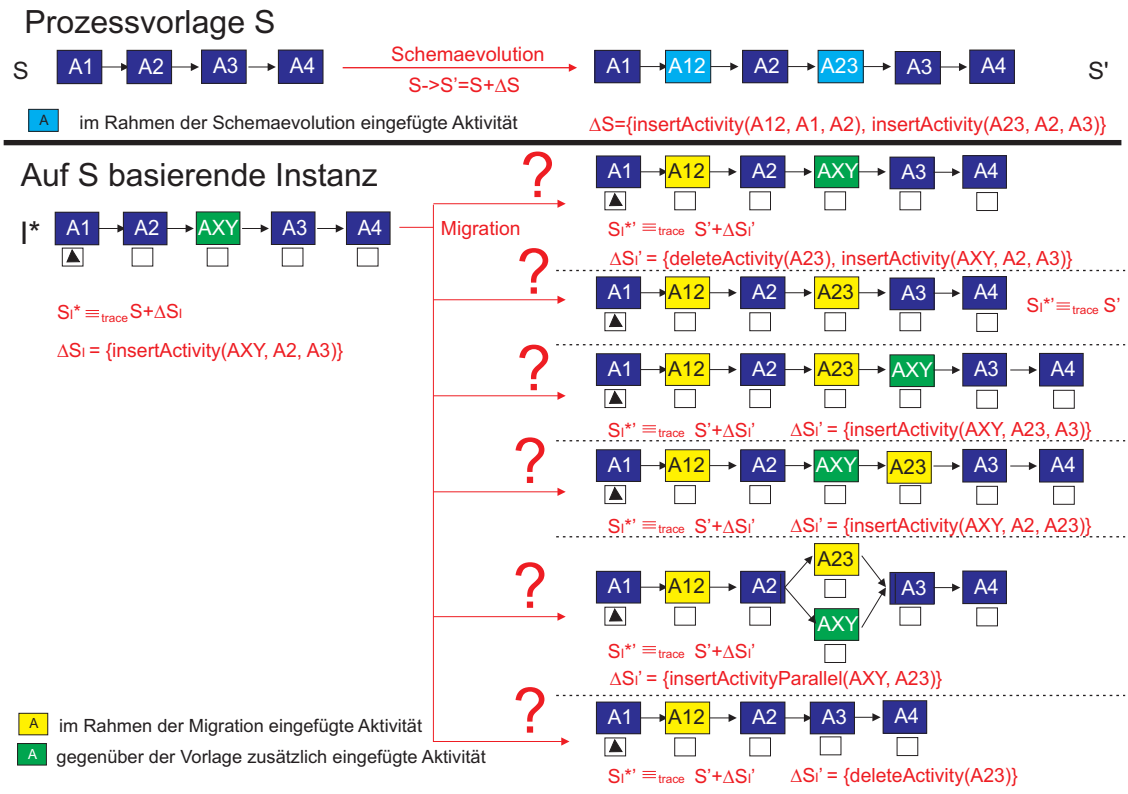

Abbildung 2.6: Konflikte bei der Migration bei partiell äquivalenten Änderungsarten

auf einer Ebene Aktivitäten gelöscht werden, auf der jeweils anderen Ebene aber dieselben Aktivitäten verschoben oder anderweitig modifiziert werden. Wiederum müssen diese Konflikte erkannt und mithilfe des Benutzers oder mit im System hinterlegten Regeln aufgelöst werden.

Ferner ist zu überprüfen, ob das aus der Migration resultierende Schema S_I*' einer verzerrten Instanz nicht gegen die in Abschnitt 2.1 angesprochenen Korrektheitskriterien für Kontroll- und Datenflüsse verstößt. Die Kombination von Instanz- und Schemaänderungen können zu strukturellen Konflikten führen, obwohl die Instanz- und die Schemaänderungen für sich genommen gegen keine Regeln verstoßen haben (vgl. [RRD03b]).

Zum Beispiel kann es in Zusammenhang mit Sync-Kanten zu Problemen kommen. Bei dem in Abbildung 2.7 dargestellten Beispiel wurde das Laufzeitschema S_I der Instanz I durch die Ad-hoc-Modifikation $\Delta S_I = \{insertSync(A22, A11)\}$ in die neue, korrekte Version S_I* überführt. Auch die neue Version S' der Prozessvorlage, die aus S durch das Einfügen der Sync-Kante zwischen A12 und A21 hervorging, d. h. $\Delta S = \{insertSync(A12, A21)\}$, genügt den Korrektheitsanforderungen. Wird jedoch nach der Migrationsregel für disjunkte Änderungen die Schemaänderung ΔS auf das dynamisch abgeänderte, korrekte Laufzeitschema S_I* für die Migration von I angewendet, so weist das resultierende Schema S_I*' einen Verstoß gegen die Korrektheits-

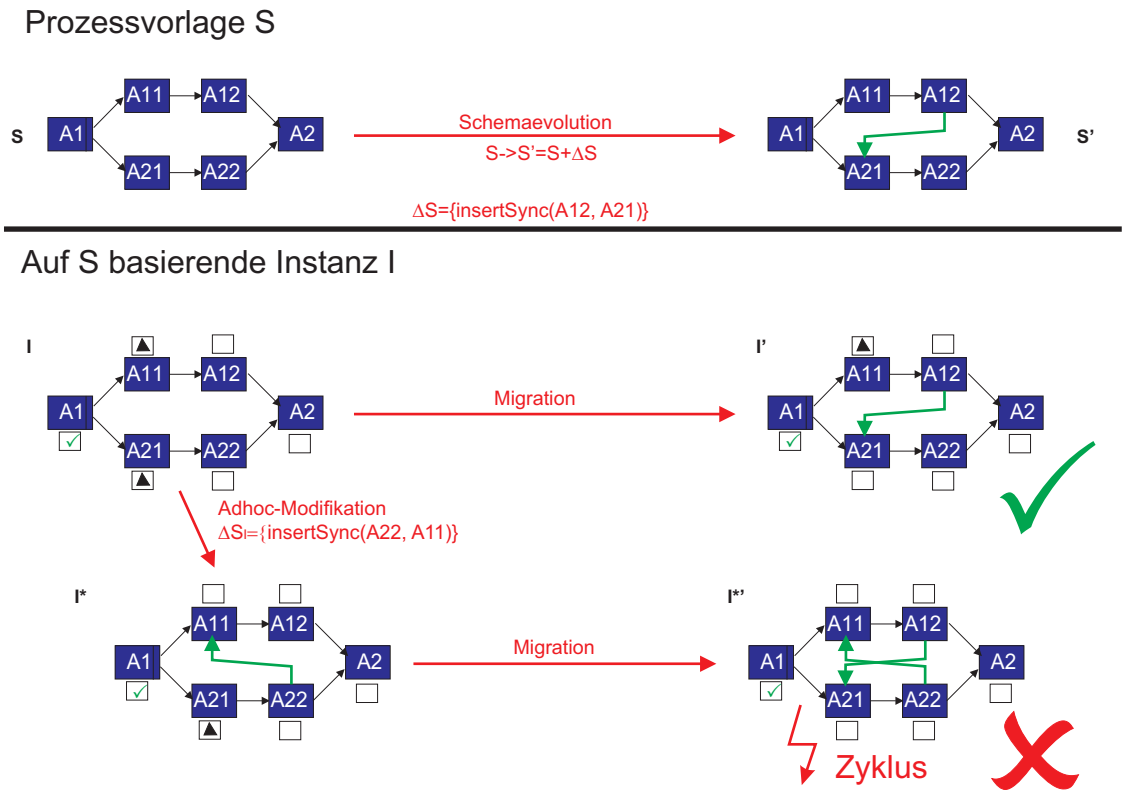

Abbildung 2.7: Inkorrekte Laufzeitschemata nach Migration verzerrter Instanzen

kriterien auf: Die Sync-Kanten bilden zusammen mit dem Kontrollfluss einen Zyklus. Dieser Zyklus führt bei der Prozessausführung zu einem Deadlock (vgl. Kapitel 2.1). Ein entsprechender Test muss verhindern, dass I auf die neue Version S' des Schemas migriert. [Ri04] beschreibt einen effizienten Test, der anhand der alten Schemaversion, der Änderungshistorie der Ad-hoc-Modifikation und der Änderungshistorie der Schemaevolution die Konflikte erkennen kann, ohne das Schema S_I*' explizit herstellen zu müssen.

2.3.3 Verträgliche und unverträgliche Instanzen

Die Migration sowohl einer unverzerrten als auch einer verzerrten Instanz ist nur zulässig, wenn die Instanz mit dem neuen Schema verträglich (engl.: *compliant*) ist [RRD03b, RRD04]. [RRD04] stellt für die verschiedenen Prozess-Modelle Korrektheitskriterien auf, welche die Verträglichkeit der Instanz garantieren. Bei Prozess-Modellen mit Ausführungshistorien lautet das Korrektheitskriterium: Eine Instanz ist mit dem neuen Schema verträglich, wenn sich ihre bisherige Ausführungshistorie (basierend auf dem alten Schema) auch auf dem neuen Schema reproduzie-

ren lässt [RRD04, Korrektheitskriterium 6]. Die Ausführungshistorie führt u. a. darüber Buch, ob und wann eine jede Aktivität gestartet und wieder beendet wurde und welche Datenelemente sie mit welchen Werten beschrieben haben bzw. welche Werte sie aus den Datenelementen gelesen haben [RRD02]. Sie gibt die ganze Vergangenheit der Instanz wieder. Daraus lässt sich der komplette Prozessablauf rekonstruieren. Wenn die Instanz auf der neuen Version der Vorlage basieren soll, dann muss diese Version auch den bisherigen Verlauf der Prozessinstanz erklären können (vgl. [CCPP98]), d. h. es muss in dem neuen Schema ein Ausführungspfad existieren, der dieselbe Historie erzeugen kann. Existiert ein solcher Pfad nicht, so kann das neue Schema nicht als Vorlage für die Instanz dienen und die Migration der Instanz darauf darf nicht erfolgen: Die Instanz ist unverträglich mit dem neuen Schema (vgl. [RRD02]).

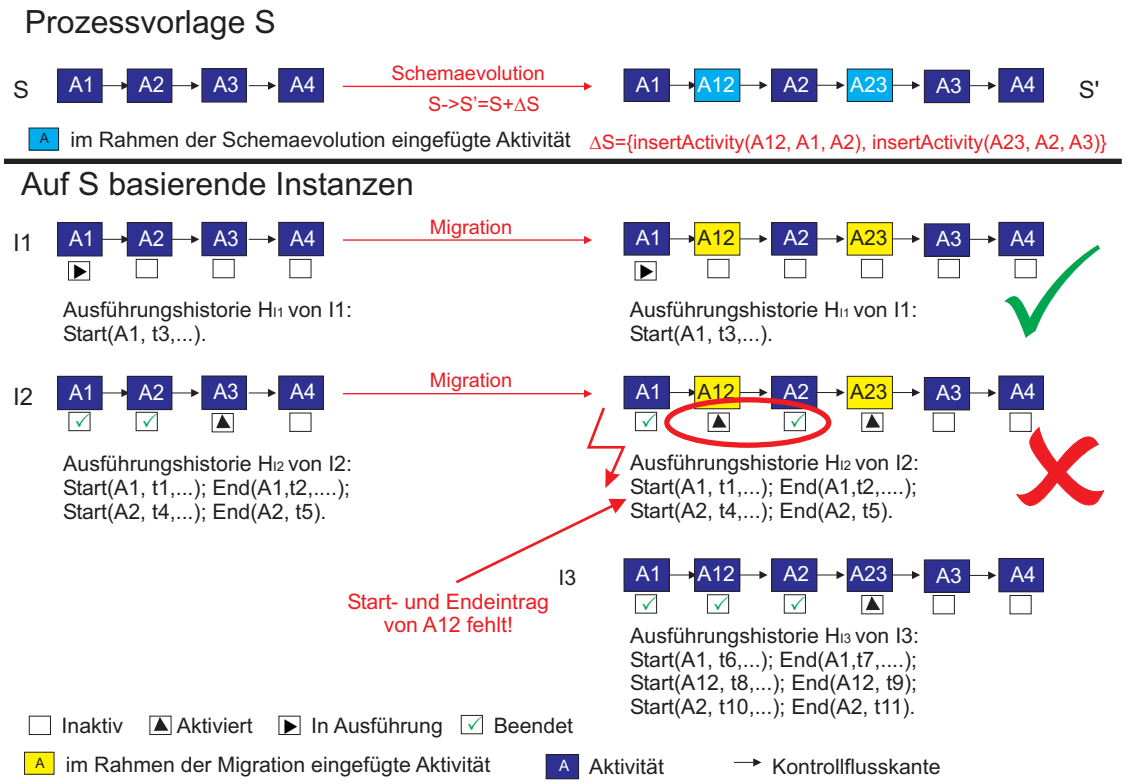

Abbildung 2.8: Ausführungshistorien und Migration

Bei der Instanz I1 aus Abbildung 2.8 wurde vor der Migration nur die Startaktivität A1 des Prozesses gestartet. Ihre Ausführungshistorie H_{I1} enthält deshalb nur den Starteintrag für diese Aktivität. A1 ist auch in der neuen Version S' der Prozessvorlage noch die Startaktivität. Somit ist bei einer Instanz, die von Anfang an nach S' abläuft, A1 die erste Aktivität, die zur Ausführung kommen kann. Folglich ist der erste Historieneintrag der Starteintrag von A1, wie die Historie H_{I3} der direkt auf S' gestarteten Instanz I3 zeigt. Die Historie H_{I1} von I1 hätte somit auch zustande kommen können, wenn die Instanz von Anfang nach dem neuen Schema

S' abgelaufen wäre. Deshalb ist Instanz I1 verträglich mit S' und die abgebildete Migration $S_{I1} \rightarrow S'_{I1}$ war zulässig.

Die Instanz I2 aus Abbildung 2.8 ist dagegen mit S' unverträglich: In der Ausführungshistorie H_{I2} von I2 folgt nach dem Start- und Endeintrag von A1 sofort der Starteintrag von A2, da nach der alten Version S der Vorlage A2 direkt nach dem Ende der Aktivität A1 ausgeführt werden kann. Diese Historie kann allerdings auf S' nicht erzeugt werden, da dabei nach A1 erst einmal A12 ausgeführt werden muss, bevor A2 gestartet werden kann. Dementsprechend muss in der Historie einer Instanz, die auf S' gestartet wurde, zwischen dem Endeintrag von A1 und dem Starteintrag von A2 definitiv noch der Start- und Endeintrag von A12 stehen (vgl. Historie H_{I3} von I3). Die Historie von I2 enthält diese Einträge nicht. I2 ist somit nicht verträglich und darf nicht migriert werden.

Wird die Migration einer unverträglichen Instanz trotzdem durchgeführt, so ist nicht nur die Vergangenheit der Instanz nicht mehr erklärbar, sondern es kann auch zu Inkonsistenzen im Prozessstatus kommen, wie in Abbildung 2.8 anhand der unverträglichen Instanz I2 gezeigt: Nach der Migration liegt die Inkonsistenz vor, dass A2 ausgeführt wurde, obwohl A12 noch nicht abgearbeitet wurde. Dies widerspricht der Regel, dass eine Aktivität erst gestartet werden kann, wenn alle Vorgängeraktivitäten, die nicht in einem abgewählten XOR-Pfad liegen, erfolgreich beendet wurden.

Das Prozess-Management-System muss anhand eines Verträglichkeitstests die zu migrierenden Instanzen auf Verträglichkeit prüfen und im Falle der Unverträglichkeit die Instanz von der Migration ausschließen. Der Test muss auf effiziente Weise die Verträglichkeit bzw. Unverträglichkeit feststellen, da in realen Anwendungsszenarien oft tausende von Instanzen gleichzeitig zur Migration anstehen können.

Das oben angegebene Korrektheitskriterium selbst liefert einen ersten Ansatz für einen Verträglichkeitstest: Die Historie muss auf dem neuen Schema reproduzierbar sein. Ein möglicher Verträglichkeitstest ist es also, die Ausführungshistorie auf dem neuen Schema Schritt für Schritt nachzuspielen (vgl. [RRD02]). Der Versuch, die Historie der unverträglichen Instanz I2 auf dem neuen Laufzeitschema S'_{I2} nachzuspielen, scheitert daran, dass nach dem Endeintrag von A1 schon der Starteintrag von A2 steht, nach der neuen Vorlage S' aber nach A1 nur A12 gestartet werden kann und nicht A2. Damit wurde I2 mit der neuen Version S' der Vorlage als nicht verträglich erkannt.

Da das Nachspielen der kompletten Historie für alle zur Migration anstehenden Instanzen u. U. sehr aufwändig sein kann, besonders wenn die Instanzen mehrere hundert Schleifendurchgänge durchlaufen haben, ist diese Art der Verträglichkeitsprüfung aus Performance-Gründen in der Praxis eher ungeeignet. Ein weiterer Nachteil ist, dass für das Nachspielen der Ausführungshistorie das neue Schema erzeugt werden muss, bevor überhaupt feststeht, ob die Instanz verträglich ist. Hinzu kommt, dass Ausführungshistorien aufgrund ihrer Größe meistens im Hintergrundspeicher gehalten werden. Der gegenüber einem Hauptspeicherzugriff langsamere Zugriff auf den Hintergrundspeicher wirkt sich zusätzlich negativ auf die Performance aus.

[RRD02] zeigt, dass die Verträglichkeit einer Instanz mit der neuen Version der Vorlage von ihrem Zustand vor der Migration abhängt. Für jede einzelne Änderungsoperation werden einige wenige, schnell überprüfbare Bedingungen an den Zustand angegeben, welche die Anwendbarkeit der Operation auf das Laufzeitschema der Instanz garantieren, so dass die bereits entstandene Ausführungshistorie reproduzierbar bleibt und keine Inkonsistenzen im Zustand der Instanz auftreten. Für das Einfügen einer Aktivität sequentiell zwischen zwei andere lautet die Bedingung, dass die Aktivität, vor die eingefügt wird, noch nicht gestartet worden sein darf. Somit führt das Einfügen einer Aktivität Y sequentiell zwischen zwei anderen Aktivitäten X und Z nicht zu Inkonsistenzen, solange Z noch nicht gestartet wurde. Die Instanz ist mit der neuen Version der Vorlage verträglich, wenn für alle Operationen der Änderungstransaktion die entsprechenden Bedingungen erfüllt sind.

Der Verträglichkeitstest läuft dann derart ab, dass für jede anzuwendende Änderungsoperation die zu untersuchenden Aktivitäten bestimmt und daraufhin deren Zustände auf Einhaltung der Bedingungen untersucht werden. Erfüllt nur eine einzige Aktivität nicht die gestellte Anforderung an ihren Zustand, so klassifiziert der Test die Instanz als unverträglich.

Der Aufwand zur Überprüfung der Bedingungen ist deutlich geringer als die ganze Ausführungshistorie nachzuspielen. Außerdem erfordert der Test auf Verträglichkeit nicht mehr den Zugriff auf die komplette Ausführungshistorie, sondern nur auf einen sehr geringen Teil daraus: Benötigt werden die Informationen, die den aktuellen Zustand betreffen. Um den Zugriff auf die Historie zu vermeiden, werden diese Informationen zusätzlich und schnell zugreifbar in Form von Zustandsmarkierungen von Aktivitäten in der Instanz hinterlegt. Der geringe Umfang dieser Daten ermöglicht es, sie im Hauptspeicher zu halten. Damit entfällt der Zugriff auf den erheblich langsameren Hintergrundspeicher[6]. Ein weiterer Vorteil ist, dass nicht das neue Laufzeitschema hergestellt werden muss, bevor die Instanz als verträglich erkannt wurde.

2.3.4 Markierungsanpassung

Auch wenn die migrierten Instanzen vom Zustand her verträglich mit den Änderungen sind, müssen im Allgemeinen an ihnen im Anschluss an die Migration noch Zustandsanpassungen vorgenommen werden, wie Abbildung 2.9 zeigt:

Bei der Instanz I ist nach der Anwendung von ΔS die Aktivität A2 noch als aktiviert markiert, obwohl die Vorgängeraktivität, die neue Aktivität A12, noch nicht abgearbeitet wurde. Sie muss wieder in den inaktiven Zustand zurückversetzt werden, d. h. die entsprechenden Arbeitsaufträge müssen aus den Arbeitslisten der infrage kommenden Bearbeiter herausgenommen werden, damit sie nicht vor A12 ausgeführt werden kann. Bei A12 liegt der umgekehrte Fall vor: Sie befindet sich fälschlicherweise noch im inaktiven Zustand, obwohl ihre direkte Vorgängeraktivität A1 bereits beendet wurde und sie nach der Vorgabe des Kontrollflusses nun eigentlich zur Ausführung anstehen müsste. Es ist damit erforderlich, ihren Zustand von “INAKTIV” in “AKTIVIERT” zu

[6] Ausnahme: Die Instanz wurde im Rahmen der regulären Speicherverwaltung auf den Hintergrundspeicher ausgelagert und muss zur Verarbeitung wieder in den Hauptspeicher geladen werden.

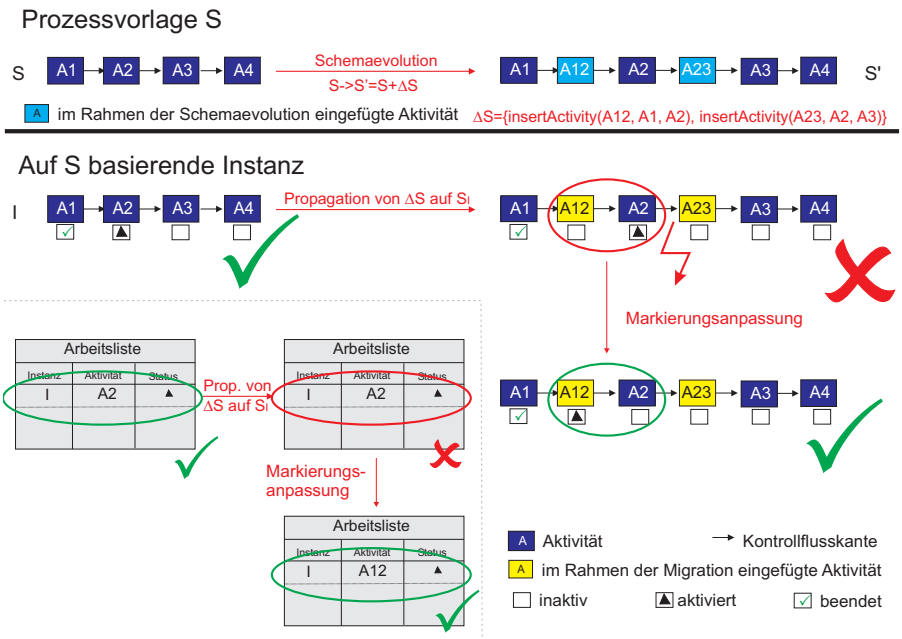

Abbildung 2.9: Die Laufzeitzustände und Arbeitslisteneinträge von I vor der Migration, nach Anwendung von ΔS aber bei noch nicht erfolgter Markierungsanpassung und nach erfolgter Anpassung.

ändern. Die relevanten Bearbeiter bekommen damit einen entsprechenden Arbeitsauftrag in ihre Arbeitsliste gestellt. Diese Anpassungen haben keinen Einfluss auf die bestehende Ausführungshistorie, da inaktive bzw. aktivierte Aktivitäten noch keine Historieneinträge geschrieben haben. Deshalb sind sie zulässig.

Welche Aktivitäten u. U. in ihrem Zustand angepasst werden müssen, hängt wiederum von den durchgeführten Änderungen ab, d. h. die Menge der anzupassenden Aktivitäten kann anhand der aufgerufenenen Änderungsoperationen bestimmt werden. [RRD02] führt die entsprechenden Regeln auf. Wird z. B. die Aktivität Y zwischen X und Z eingefügt, so muss die Nachfolgeaktivität Z von Y auf „INAKTIV“ zurückgesetzt werden, wenn sie sich bereits in dem Zustand „AKTIVIERT“ befand. Liegt der Status „INAKTIV“ oder „ABGEWÄHLT“ vor, so sind keine Anpassungen vorzunehmen[7]. Eine Aktivität bekommt den Status „ABGEWÄHLT“, wenn sie in einem abgewählten Pfad einer XOR-Verzweigung liegt. Zusätzlich muss die eingefügte Aktivität Y noch aktiviert werden, wenn die Voraussetzungen dafür gegeben sind, bzw. als „ABGEWÄHLT“ markiert werden, wenn sie in einen abgewählten Pfad einer XOR-Verzweigung eingefügt wurde. Ein effizienter Algorithmus für die Markierungsanpassung wird ebenso in [RRD02] beschrieben.

[7]Einen anderen Zustand als „AKTIVIERT“, „INAKTIV“ oder auch „ABGEWÄHLT“ kann Z direkt nach der Migration nicht innehaben, da die Instanz sonst nicht verträglich mit dieser Operation gewesen wäre und es somit nicht zu einer Migration dieser Instanz gekommen wäre (vgl. Abschnitt 2.3.3).

2.3.5 Ablauf der Migration einer Instanz

Zusammengefasst kann damit der Migrationsprozess in sechs Phasen eingeteilt werden, wie das Zustandsdiagramm aus Abbildung 2.10 zeigt. In der ersten Phase, der Verträglichkeitsprüfung,

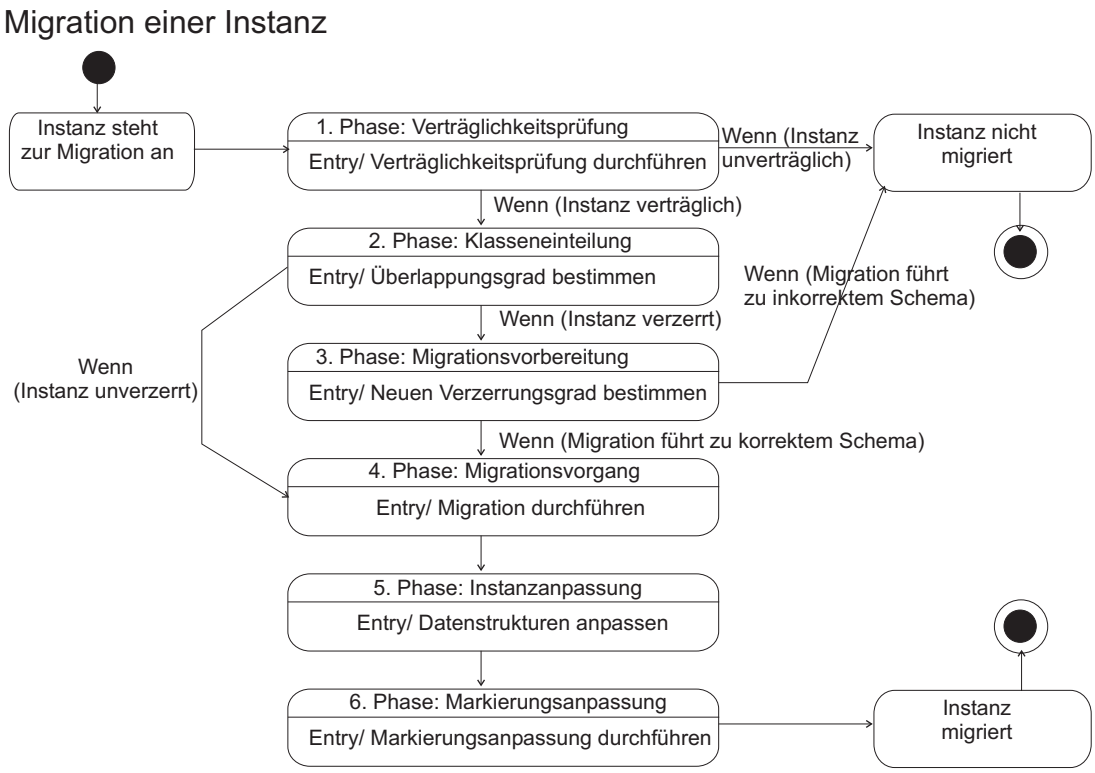

Abbildung 2.10: Das Zustandsdiagramm zur Migration.

wird eine zur Migration anstehende Instanz auf Verträglichkeit mit den Änderungen geprüft. Im Falle der Unverträglichkeit wird die Instanz von der Migration ausgeschlossen. In der zweiten Phase wird sie einer der beiden Klassen „Verzerrte Instanz“ und „Unverzerrte Instanz“ zugeordnet. Bei einer verzerrten Instanz wird zusätzlich der Überlappungsgrad der Instanzänderung mit der Schemaänderung bestimmt. Durch die Klasseneinteilung wird deren Migrationsstrategie festgelegt. In der darauf folgenden Phase werden die zusätzlich für die Migration einer verzerrten Instanz benötigten Informationen berechnet, wie z. B. der neue Bias, mit dem die Instanz nach der Migration gegenüber der neuen Vorlage verzerrt sein wird. Unverzerrte Instanzen können diese Phase überspringen, da keine weiteren Informationen benötigt werden. Zusätzlich wird in dieser Phase untersucht, ob das resultierende Schema Verstöße gegen die Korrektheitskriterien des zugrunde gelegten Prozess-Modells aufweisen wird. Im Falle eines Verstoßes wird die Instanz an dieser Stelle von der Migration zurückgewiesen. Stehen alle für die Migration benötigten Informationen zur Verfügung, so tritt die Instanz in die vierte Phase der Migration ein, in der

die bisher unverzerrten und verzerrten Instanzen gemäß der in Phase 2 festgelegten Migrationsstrategie an die neue Schemaversion angepasst werden. Wenn im Rahmen der Migration neue Aktivitäten dazu kamen, so müssen die Datenstrukturen angepasst werden, welche die Laufzeitinformationen, wie z. B. den momentanen Ausführungsstand, speichern, um entsprechende Einträge für diese Aktivitäten aufnehmen zu können. Dafür ist die fünfte Phase, die Instanzanpassung, gedacht. Andere Änderungsoperationen fordern andere Anpassungen. Der Migrationsprozess endet mit der Phase der Markierungsanpassung, in der die Zustände von bestimmten Aktivitäten der Instanz, wie in Abschnitt 2.3.4 beschrieben, angepasst werden.

2.4 Dynamische Änderungen

Verträglichkeitsprüfung, Instanzanpassung und Markierungsanpassung spielen auch bei dynamischen Änderungen von Instanzen zur Laufzeit eine Rolle.

Wie schon angedeutet wird durch eine dynamische Änderung das Laufzeitschema S_I einer Instanz I gegenüber der Vorlage S durch Änderungsoperationen ΔS_I mehr oder weniger stark abgeändert. Für das resultierende Laufzeitschema S_I* der nun gegenüber der Vorlage verzerrten Instanz I gilt: $S_I* = S_I + \Delta S_I \equiv_{trace} S + \Delta S_I$.

Anmerkung: Eine Instanz kann auch innerhalb mehrerer Änderungstransaktionen abgeändert worden sein. Nach der i-ten Transaktion gilt dann für das neue Laufzeitschema S_I*, wenn dazwischen keine Migration stattgefunden hat:
$S_I* = S_I*_i = S_I*_{i-1} + \Delta S_{Ii} = S_I + \sum_{j=1}^{i} \Delta S_{Ij} \equiv_{trace} S + \sum_{j=1}^{i} \Delta S_{Ij}$.

Nach einer Instanzänderung muss bei der Instanz nicht nur wieder statische Korrektheit vorliegen, sondern auch die dynamische (vgl. [Re00]). Statische Korrektheit liegt vor, wenn wie im Falle der Schemaevolution das resultierende Laufzeitschema der Instanz gegen keine Korrektheitskriterien des Prozess-Modells in Bezug auf den Aufbau eines Schemas verstößt. Zum Beispiel darf das Laufzeitschema in ADEPT keine Zyklen bezüglich Kontrollfluss- und Sync-Kanten aufweisen. Dynamische Korrektheit liegt vor, wenn keine Inkonsistenzen im Zustand der Instanz existieren. Somit dürfen die Instanzänderungen zu keinen Zustandsinkonsistenzen führen, die nicht durch die von der Migration her bekannte Markierungsanpassung aufgelöst werden dürfen. In ADEPT darf z. B. keine Aktivität sequentiell vor eine bereits ausgeführte Aktivität eingefügt werden, da sonst gegen die Regel verstoßen wird, dass eine Aktivität erst gestartet werden darf, wenn die Vorgänger-Aktivitäten bereits beendet wurden. Änderungen führen nicht zu nicht durch die Markierungsanpassung bereinigbaren Zustandsinkonsistenzen, wenn sie verträglich mit dem aktuellen Zustand der Instanz sind. Deshalb muss vor der Anwendung einer Änderungsoperation deren Verträglichkeit überprüft werden. Im Falle einer Unverträglichkeit muss die Operation zurückgewiesen werden.

Wie bei der Migration muss nach einer Änderung der Instanz die Datenstruktur, welche die Zustände aller Aktivitäten speichert, nach der Modifikation an das geänderte Schema angepasst

werden, um z. B. den Zustand einer neu eingefügten Aktivität aufnehmen zu können. Dies geschieht im Rahmen der aus der Migration bekannten Instanzanpassung.

Ebenso kann eine Instanzänderung die Markierungsanpassung notwendig machen. Wird z. B. eine Aktivität Y sequentiell direkt vor die aktivierte Aktivität Z eingefügt, dann muss Z wieder deaktiviert werden, um deren Ausführung vor Y zu verhindern. Y muss stattdessen aktiviert werden, sofern nichts anderes dagegen spricht, wie z. B. dass Y aufgrund einer Sync-Kante auf das Ende einer anderen Aktivität warten muss.

Um dem Benutzer immer eine korrekte Sicht auf die Instanz bieten zu können, wird der Ablauf vorgeschlagen, der sich mit dem Zustandsdiagramm aus Abbildung 2.11 darstellen lässt:

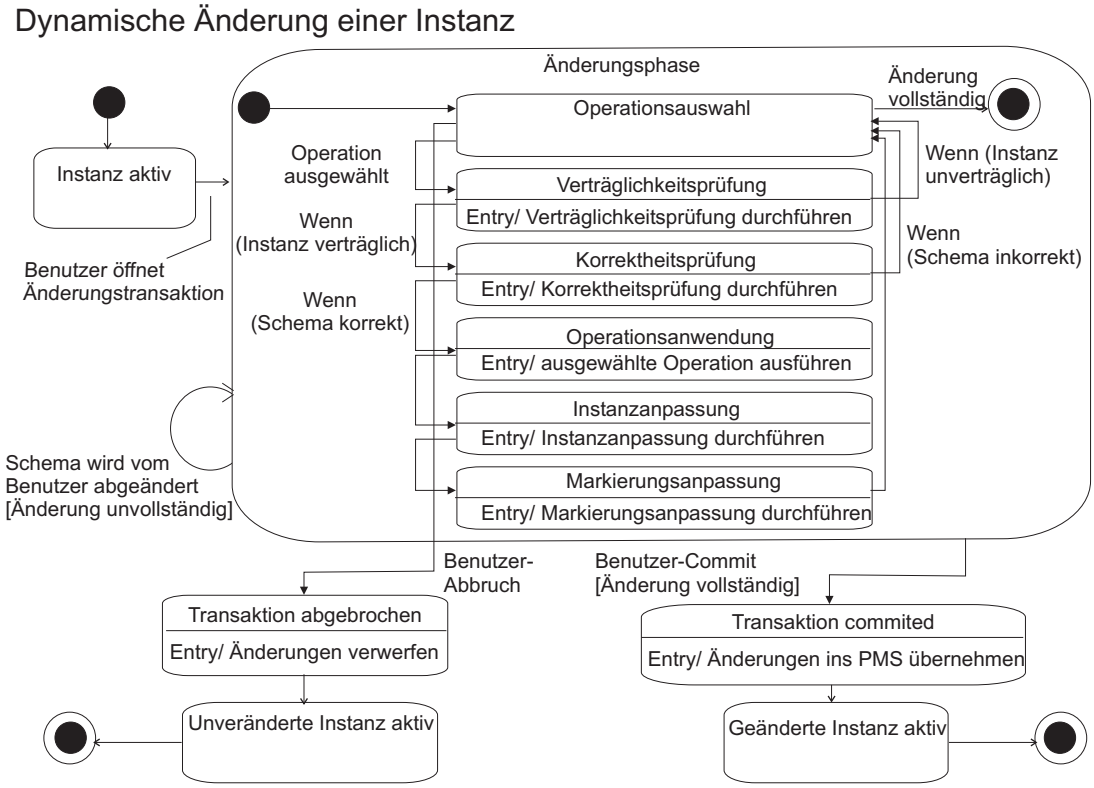

Abbildung 2.11: Das Zustandsdiagramm zur dynamischen Änderung einer Instanz.

Um eine Änderung durchzuführen, wird mit dem Beginn der Änderungstransaktion die zu ändernde Instanz auf den Client-Rechner kopiert. Die Instanz tritt damit in die Änderungsphase ein. Der Benutzer kann dann mit einem entsprechenden Tool die Änderungen am Laufzeitschema vornehmen. Bevor jedoch eine gewünschte Operation auf das Laufzeitschema angewendet wird, wird überprüft, ob die Operation vom aktuellen Laufzeitzustand her überhaupt erlaubt werden darf. Dafür kommen die gleichen Regeln wie bei der Verträglichkeitsprüfung bei der

Migration zur Anwendung (vgl. Abschnitt 2.3.3). Fällt der Test negativ aus, so wird dem Benutzer diese Operation verweigert. Die Verweigerung führt systemseitig nicht automatisch zum Abbruch der Änderungstransaktion. Der Benutzer kann innerhalb des gleichen Transaktionskontextes weitere Modifikationen durchführen. Fällt der Test positiv aus, so wird die Operation auf das Laufzeitschema angewendet, vorausgesetzt, das resultierende Schema widerspricht nicht den Korrektheitskriterien des zugrunde gelegten Prozess-Modells. Danach werden sofort – wenn nötig – die Datenstrukturen, welche die aktuellen Zustände speichern, analog zur Phase 5 bei der Migration angepasst und die Markierungszustände analog zur Phase 6 bei der Migration aktualisiert. Erst danach kann der Benutzer weitere Änderungsoperationen durchführen. Bei einem Commit der Änderungstransaktion werden die Änderungen auf den PMS-Server übernommen. Für einen Abbruch der Transaktion wird die Kopie einfach verworfen.

Auf die Probleme, die auftreten können, wenn zeitgleich zur Ad-hoc-Modifikation die Instanz weitergeschaltet wird, und wie diese Probleme verhindert werden können, wird in Kapitel 5.4 näher eingegangen.

Eine andere mögliche Vorgehensweise für das Abändern einer Instanz ist, den Benutzer zuerst sämtliche Modifikationen auf dem Laufzeitschema der Instanz ohne Prüfungen durchführen zu lassen und erst später im Rahmen der Commit-Behandlung die notwendige Verträglichkeitsprüfung, Instanzanpassung und Markierungsanpassung vorzunehmen. Im Falle der Unverträglichkeit muss der Benutzer dann aber seine Modifikationen überarbeiten, was aufwändig sein kann.

2.5 Zusammenfassung

Dieses Kapitel zeigt, dass Änderungen des Schemas nicht einfach auf die Instanzen übernommen werden dürfen, da es im Falle von unverträglichen Instanzen oder bei entsprechender Kombination von Instanz- und Schemaänderungen zu Inkonsistenzen im Zustand oder zu Verstößen gegen die Korrektheitskriterien des zugrunde liegenden Prozess-Modells kommen kann. Deshalb müssen vor der Migration der Instanzen Tests sicherstellen, dass es dazu nicht kommt. Weiter zeigt dieses Kapitel, dass die Art und Weise, wie Instanzen an die neue Vorlage anzupassen sind, davon abhängt, ob eine unverzerrte Instanz oder eine verzerrte Instanz vorliegt und im Falle einer verzerrten Instanz, inwieweit sich die Instanz- und Schemaänderungen überlappen.

Wie die Schemaevolution, die dynamische Änderung einer Instanz und die Phase 4 des Migrationsprosses physisch realisiert werden, hängt davon ab, wie Prozessvorlagen und -instanzen im Speicher repräsentiert sind. Das nächste Kapitel leitet eine Repräsentationsform her und beschreibt darauf aufbauend die physische Realisierung der in diesem Kapitel beschriebenen Mechanismen der Adaptivität.

Kapitel 3

Realisierung von Prozessvorlagen und -instanzen

In diesem Kapitel wird Schritt für Schritt eine Architektur erarbeitet, mit der sowohl Ad-hoc-Modifikationen von Instanzen als auch Änderungen von Prozessvorlagen durchgeführt werden können. Zudem unterstützt dieser Ansatz die Migration von unverzerrten sowie von verzerrten Instanzen auf ein modifiziertes Prozessschema, ohne dabei das System stark zu belasten oder einen hohen Ressourcenbedarf aufzuweisen. Die Realisierungsentscheidungen werden u. a. durch Gegenüberstellung mit anderen Realisierungsvarianten begründet.

3.1 Naiver Architekturansatz

Der naive Ansatz, Instanzen zu repräsentieren, zeigt Abbildung 3.1. Er besteht darin, für jede Instanz ein Objekt anzulegen und darin sowohl die Laufzeitinformationen, wie z. B. den Ausführungszustand der Instanz, als auch die Prozessbeschreibungen, wie den Kontroll- und Datenfluss, zu speichern.

Die Anpassung unverzerrter Instanzen an eine neue Version der zugrunde liegenden Prozessvorlage erfolgt, indem die Änderungsoperationen, welche die Vorlage von der alten in die neue Version überführt haben, auf die in den entsprechenden Instanz-Objekten hinterlegten Schemata erneut angewendet werden. Bei verzerrten Instanzen muss eine dem Überlappungsgrad von Schema- und Instanzänderung entsprechend angepasste Operationenmenge angewendet werden. Bei verzerrten Instanzen, wie z. B. bei der Instanz I3 aus Abbildung 3.1, weicht das im Instanz-Objekt hinterlegte Schema von der Vorlage ab.

Der große Nachteil bei diesem Ansatz ist, dass dabei die Prozessbeschreibung unnötig oft redundant gespeichert wird: Jede (unverzerrte) Instanz desselben Prozesstyps läuft nach dem gleichen Schema ab. Deshalb sind in jedem der entsprechenden Instanz-Objekte dieselben Schemain-

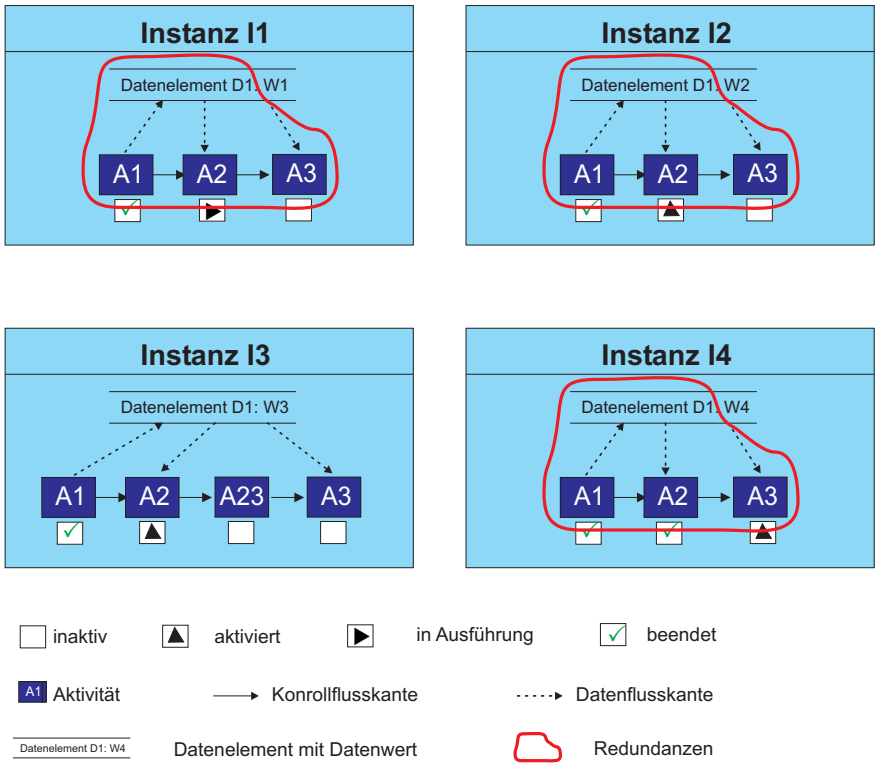

Abbildung 3.1: Naiver Ansatz, Instanzen zu repräsentieren

formationen – die Prozessbeschreibung – enthalten. Es lässt sich erheblich Speicher einsparen, wenn die redundanten Informationen aus den jeweiligen Instanz-Objekten herausgenommen und stattdessen einmal in einem speziellen, dem Prozesstyp zugeordneten Objekt hinterlegt werden. Jede Instanz des entsprechenden Prozesstyps greift dann auf dieses Objekt zu, um an die Informationen über das Prozessschema zu kommen. Der im Folgenden vorgestellte Architekturansatz beschreitet diesen Weg.

3.2 Architekturansatz

Ein in der Literatur (z. B. [We01]) vorgeschlagener Implementierungsansatz für Prozessvorlagen und Instanzen wird in Abbildung 3.2 illustriert. Bei diesem Ansatz wird die Prozessbeschreibung in einem Objekt, der *Prozessvorlage*, gekapselt, das den Prozesstyp repräsentiert. Die *Instanz-Objekte*, welche Prozessinstanzen repräsentieren, enthalten selber nur Laufzeitinformationen, wie den Ausführungsstatus von Aktivitäten und logisch, nicht unbedingt physisch, den Inhalt der Datenelemente. Die Prozesstypzugehörigkeit wird durch eine Referenz auf das entsprechende

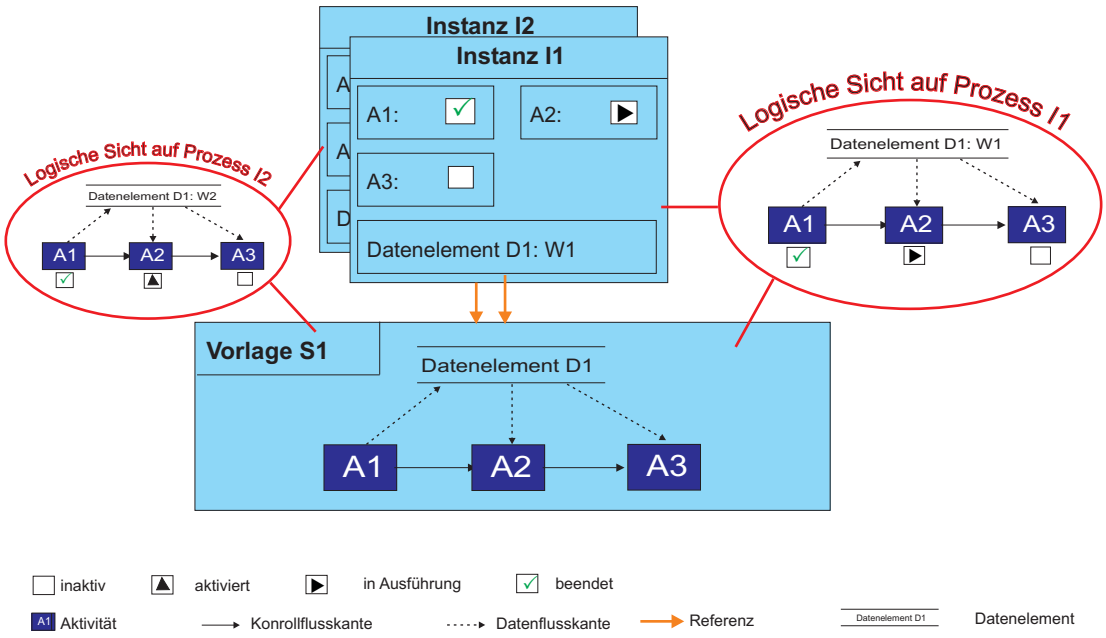

Abbildung 3.2: Die Repräsentation von Instanzen eines Prozesstyps

Prozessvorlagen-Objekt ausgedrückt. Alle Instanzen eines Prozesstyps referenzieren dasselbe Vorlagen-Objekt. Dieser Ansatz soll als Ausgangsbasis dienen.

Gegenüber der Variante, für jede Instanz die jeweilige Prozessbeschreibung redundant zu speichern (siehe Abschnitt 3.1), resultiert ein erheblich geringerer Speicherplatzbedarf bei großer Anzahl von Instanzen. Ein weiterer Vorteil ist, dass sich damit auch signifikant Rechenzeit einsparen lässt, da strukturverändernde Operationen bei einer Schemaevolution nur einmal auf die Prozessvorlage angewendet werden müssen, bei der anderen Variante dagegen auf jede einzelne Prozessinstanz.

3.3 Schemaevolution und Migration unverzerrter Instanzen

Bei dieser Implementierung darf allerdings die Schemaänderung nicht direkt auf dem Vorlagen-Objekt ausgeführt werden, welches die aktuelle Version repräsentiert. Die Propagation der Schemaänderungen auf Instanzen würde zwar damit in einem Vorgang erfolgen, aber dann können auch Instanzen fälschlicherweise migriert werden, die für diese Änderungen zu weit fortgeschritten sind. In Abbildung 3.3 laufen Instanz I1 und Instanz I2 auf derselben Vorlage S1, wobei I1 weiter fortgeschritten ist als I2. Da sich A2 bei I1 bereits in Ausführung befindet, ist I1 unverträglich mit der neuen Version des Schemas S1, bei der vor A2 die Aktivität A12 eingefügt wird. Wird nun die Änderung direkt auf S1 ausgeführt, so migriert zwar I2 ordnungsgemäß, aber I1

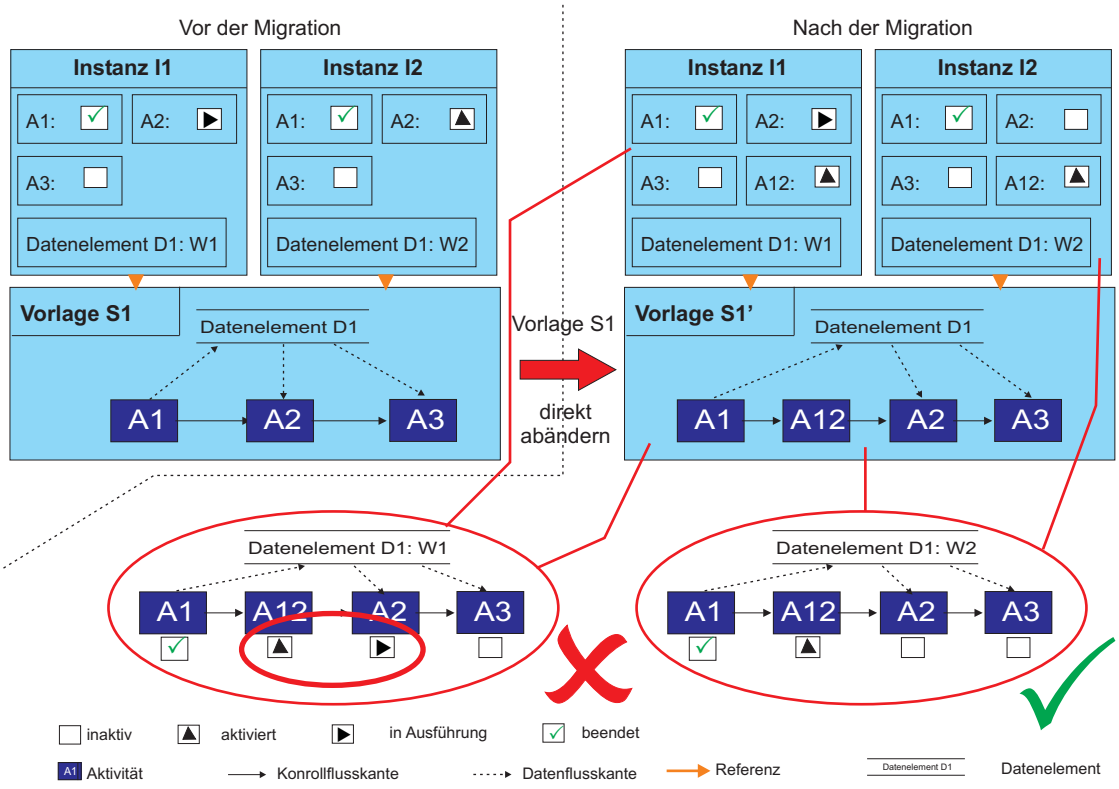

Abbildung 3.3: Migration durch direkte Abänderung der Vorlage

folgt verbotenerweise ebenfalls dem neuen Schema, da sie weiterhin das gleiche Vorlagen-Objekt referenziert wie I2. Eine Inkonsistenz ist die Folge: A2 von Instanz I1 wurde gestartet, bevor die Vorgängeraktivität A12 ausgeführt worden ist.

Die Koexistenz von Instanzen, die auf verschiedenen Versionen der Prozessvorlage laufen, kann sichergestellt werden, indem eine Kopie des Prozessvorlagen-Objekts angelegt wird, daran die Modifikationen ausgeführt werden und dann alle migrierbaren Instanzen auf dieses Objekt umgehängt werden. In diesem Fall besteht der Vorgang des Migrierens aus dem „Umbiegen“ der Schemareferenz auf die neue Version.

3.4 Repräsentationsvariante für verzerrte Instanzen

Wie lassen sich mit obigen Implementierungsvorschlag instanzbezogene Ad-hoc-Modifikationen abbilden? Eine Möglichkeit ist es, eine Kopie des entsprechenden Prozessvorlagen-Objekts anzufertigen, die Änderungen darauf anzuwenden und dann die Instanz auf dieses Schema umzubiegen (siehe Abbildung 3.4).

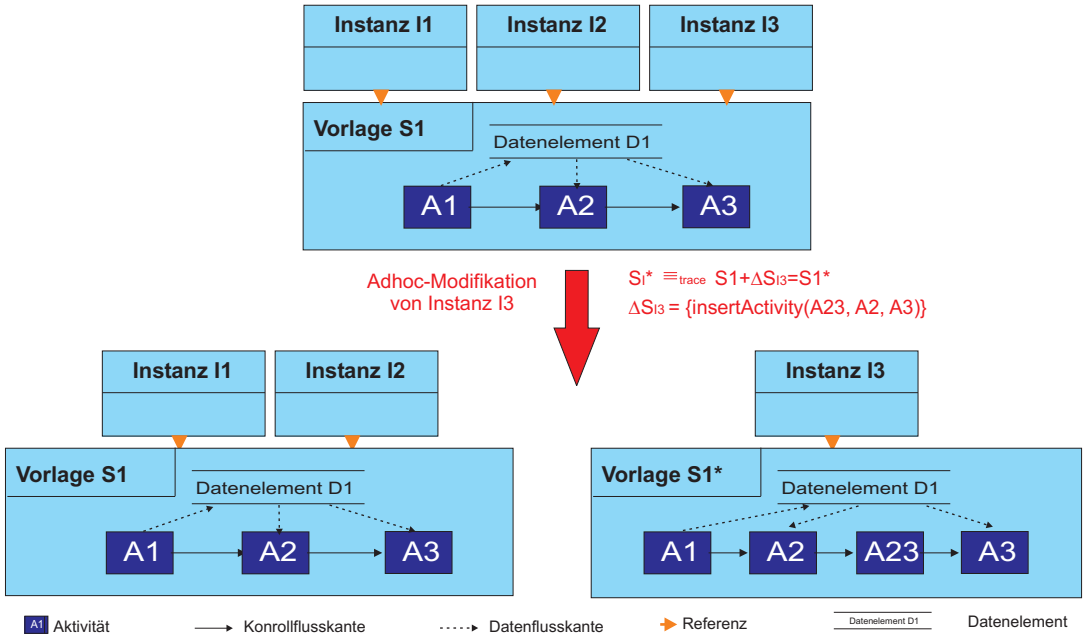

Abbildung 3.4: Fehlende Typ-Zugehörigkeit nach einer dynamischen Änderung

Von Nachteil ist, dass damit die ursprüngliche Prozesstypzugehörigkeit verloren geht: Die verzerrte Instanz I3 referenziert nicht mehr das ursprüngliche Vorlagen-Objekt S1, obwohl sie noch von dem durch dieses Objekt beschriebenen Prozesstyp abstammt. Ohne zusätzliche Vorkehrungen wird die Instanz I3 bei einer späteren Schemaevolution nicht mehr berücksichtigt. Hinzu kommt, dass dieser Ansatz zu der anfangs in Abschnitt 3.1 erwähnten Lösung, bei der in jeder Instanz der gesamte Prozessablauf hinterlegt ist, degeneriert, wenn mehr und mehr Instanzen abgeändert werden.

Berücksichtigt man aber, dass bei Instanzänderungen im Allgemeinen nur ein kleiner Teil des ursprünglichen Prozessschemas angepasst wird, so lässt sich eine alternative Strategie zur Repräsentation von verzerrten Instanzen anwenden: Man speichert bei jeder modifizierten Instanz nur die Abweichungen vom ursprünglichen Schema. Diese können auf zweierlei Arten repräsentiert werden, entweder durch die Änderungsoperationen selbst oder durch eine Delta-Schicht, die im folgenden Abschnitt 3.5 beschrieben wird.

Abweichungen durch die Änderungsoperationen selbst zu repräsentieren, kann aufgrund der kompakten Darstellungsmöglichkeit einer Änderungsoperation erheblich Speicher gegenüber der zweiten Methode einsparen. Es muss aber zur Beantwortung einer jeden Schemaanfrage, wie z. B. „Gib mir die Nachfolgeraktivität von X“ erst temporär das neue Schema durch Anwendung der Änderungsoperationen auf eine Kopie der Vorlage hergestellt werden, was erhebliche Performance-Einbußen bedeutet.

Deshalb ist es besser, das geänderte Schema sofort zugriffsbereit zu haben, wie es bei der Delta-Schicht-Methode der Fall ist.

3.5 Die Delta-Schicht

Abbildung 3.5 verdeutlicht das Konzept der Delta-Schicht. Die Delta-Schicht wird durch ein

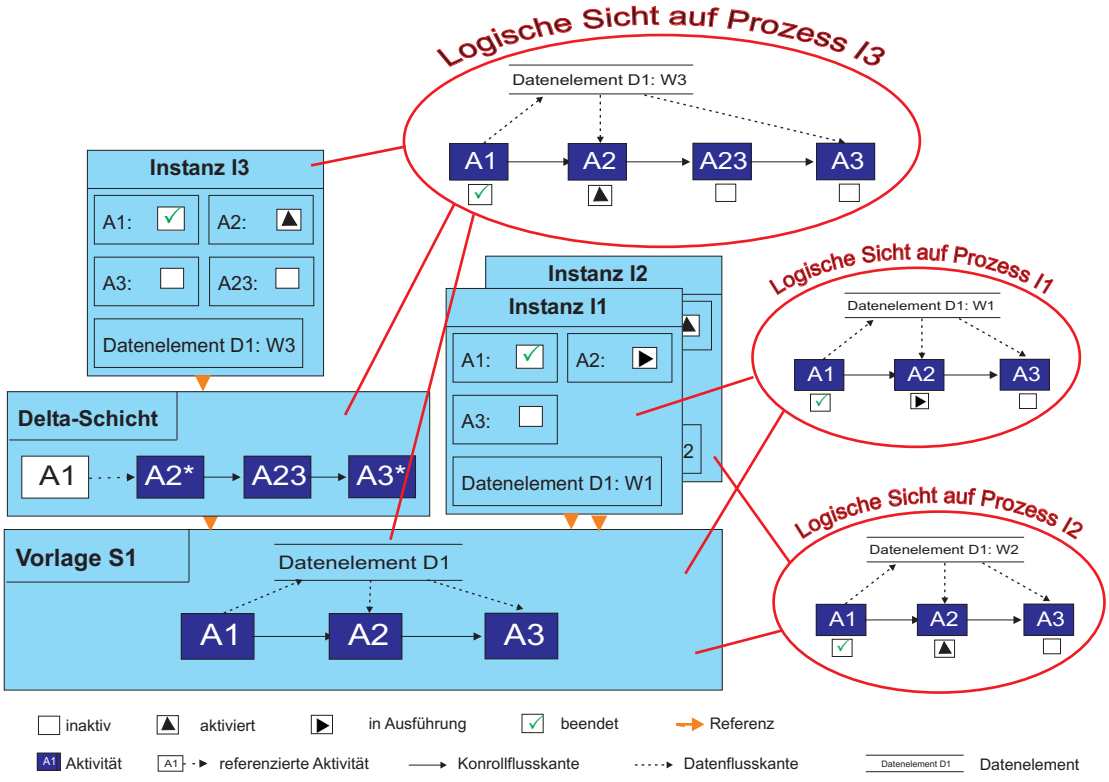

Abbildung 3.5: Das Konzept der Delta-Schicht

Objekt repräsentiert, das die gleichen Schnittstellen wie das Prozessvorlagen-Objekt besitzt und daher nach außen hin die gleichen Operationen anbietet. Der Unterschied zu dem originalen Vorlagen-Objekt besteht darin, dass es nicht den gesamten Prozessgraphen widergibt, sondern nur die Ausschnitte, die durch instanzbezogene Modifikationen geändert wurden. Daher der Name *Delta-Schicht*.

Zusammen mit dem Prozessvorlagen-Objekt, das den Prozesstyp der Instanz bestimmt, repräsentiert das Delta-Schicht-Objekt das gegenüber der Prozessvorlage abgeänderte Laufzeitschema der modifizierten Instanz. Das Instanz-Objekt, das die geänderte Instanz repräsentiert, referenziert dann nicht mehr, wie in Abbildung 3.5 anhand von Instanz I3 gezeigt, das entsprechende

Vorlagen-Objekt, sondern das Delta-Schicht-Objekt. Erst das Delta-Schicht-Objekt setzt auf dem originalen Vorlagen-Objekt auf und bewahrt auf diesem Weg u. a. die Typzugehörigkeit der Instanz I3 zum Prozesstyp S1. Die unveränderten Instanzen, wie z. B die Instanzen I1 und I2, referenzieren das originale Prozessvorlagen-Objekt weiterhin direkt. Bei modifizierten Instanzen werden deshalb Anfragen, wie z. B. "Gib mir alle direkten Nachfolgeaktivitäten von Aktivität Z!", zuerst an die Zwischenschicht gerichtet, bei unveränderten dagegen direkt an das Prozessvorlagen-Objekt.

Aus Gesichtspunkten der Speicherverwaltung wäre es sinnvoll, alle Instanz-Objekte verzerrter Instanzen, die in ihrem Laufzeitschema ausführungsäquivalent sind, dasselbe Delta-Schicht-Objekt referenzieren zu lassen, anstatt für jedes ein extra Delta-Schicht-Objekt anzulegen. Es würde allerdings einen erheblichen Aufwand kosten, bei jeder Ad-hoc-Modifikation einer Instanz genau das Delta-Schicht-Objekt aus den eventuell hundert bestehenden herauszusuchen – sofern überhaupt schon vorhanden –, das exakt dieselben Auswirkungen auf das zugrunde gelegte Prozessschema verursacht, wie mit der Ad-hoc-Modifikation beabsichtigt ist. Aus diesem Grund ist für die Praxis die Variante vorzuziehen, bei der jeweils ein Delta-Schicht-Objekt genau einer Instanz zugeordnet ist.

Da sich nach außen hin ein Delta-Schicht-Objekt nicht von einem Vorlagen-Objekt unterscheidet, kann ein Delta-Schicht-Objekt statt einem Vorlagen-Objekt auch wieder ein Delta-Schicht-Objekt referenzieren, d. h. mehrere Delta-Schicht-Objekte können übereinander gestapelt werden. Es muss nur sichergestellt sein, dass stets die unterste Delta-Schicht das typ-bestimmende Vorlagen-Objekt referenziert. Alle Delta-Schicht-Objekte zusammen bilden dann das gegenüber der Vorlage abgeänderte Laufzeitschema. Zum Beispiel können temporäre Änderungen[1], die nur einige Schleifendurchläufe lang Gültigkeit haben, in einer gesonderten Schicht gespeichert werden. Temporäre Änderungen rückgängig zu machen besteht dann nur aus dem Verwerfen des entsprechenden Delta-Schicht-Objekts[2].

In diesem Zusammenhang kommt die Frage auf, warum bei der Schemaevolution die Änderungen von einer Version auf die nächste nicht durch Delta-Schicht-Objekte repräsentiert werden, anstatt bei jedem Versionswechsel das Objekt der alten Vorlagenversion komplett zu kopieren und darauf die Änderungen durchzuführen (siehe Abschnitt 3.3). Der Grund hierfür ist, dass bei der Verwendung von Delta-Schicht-Objekten meistens der Zugriff sowohl auf ein oder mehrere Delta-Schicht-Objekte als auch auf das originale Vorlagen-Objekt notwendig ist, um an die aktuell geltenden Schemainformationen zu kommen, wie im Folgenden beschrieben wird. Bei Vorlagen, die u. U. von hunderten Instanzen referenziert werden und auf die bei jedem Weiterschalten von Aktivitäten zugegriffen werden muss, bedeutet der Mehrfachzugriff eine zu große Performance-Einbuße.

Wie die Delta-Schicht selbst realisiert werden kann, hängt von der Repräsentation der Knoten und Kanten des Prozessgraphen ab. In unserer Implementierung (siehe Abschnitt 3.8) existieren

[1] Siehe dazu [We97]

[2] Es muss dabei nur sichergestellt werden, dass es nicht zu Inkonsistenzen kommt, wenn sich andere Modifikationen auf diese temporären Änderungen bezogen haben.

keine Kanten-Objekte. Wir speichern zu jeder Aktivität die ID der Vorgänger- und Nachfolgeraktivitäten und stellen dadurch implizit den Kontrollfluss her. Vor einer dynamischen Änderung werden alle Aktivitäten-Objekte automatisch in die Delta-Schicht kopiert, die von der Änderung betroffen sind. Die Änderungen werden dann auf den Kopien durchgeführt.

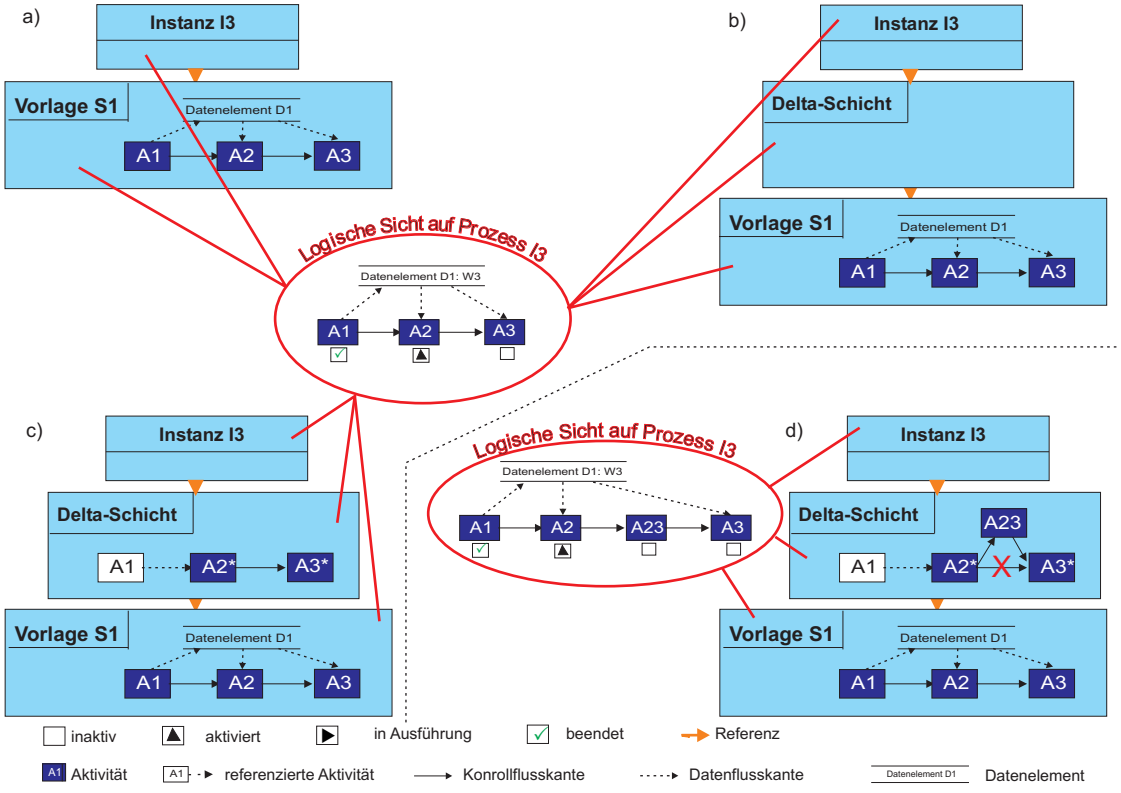

Abbildung 3.6: Serielles Einfügen einer Aktivität infolge einer Ad-hoc-Modifikation

Um z. B. die Aktivität A23 sequentiell zwischen die beiden Nachbaraktivitäten A2 und A3 in die Instanz I3 aus Abbildung 3.6 a einzufügen, werden zuerst die Aktivitäten-Objekte A2 und A3 komplett, inklusive der ID, kopiert (Abbildung 3.6 c) und als A2*[3] und A3* in das leere, zuvor zwischen das Instanz- und Vorlagen-Objekt gehängte Delta-Schicht-Objekt (Abbildung 3.6 b) eingefügt. A2 und A3 müssen kopiert werden, da sich ihre Nachfolger- bzw. Vorgängermengen ändern. Danach wird das Aktivitäten-Objekt A23 in der Delta-Schicht angelegt. Schließlich wird die Aktivität A23 sequentiell zwischen A2* und A3* eingereiht, indem die Vorgänger-Nachfolger-Beziehungen gesetzt werden (Abbildung 3.6 d): Die ID von A23 wird anstelle der von A3 in die Liste der Nachfolger von A2* und anstelle der ID von A2 in die Liste der Vorgänger von A3* aufgenommen. A23 speichert analog dazu die ID von A2* in seiner Vorgängerliste und die ID von A3* in seiner Nachfolgerliste. In einer Implementierung mit Kanten-Objekten müssen für

[3]Der Stern soll nur zum besseren Verständnis beitragen.

die Einfüge-Operation nicht A2 und A3 in die Delta-Schicht kopiert werden. Es werden nur A23 und die Kantenobjekte für die Verbindung A2→A23[4] und A23→A3 angelegt. Zusätzlich muss A2→A3 als gelöscht markiert werden.

Soll eine Aktivität gelöscht werden, so muss bei beiden Ansätzen die entsprechende Aktivität in der Delta-Schicht durch eine Null-Aktivität ersetzt werden oder anderweitig als gelöscht markiert werden. Null-Aktivitäten sind Aktivitäten, denen keine Funktionen zugeordnet sind und die nach der Aktivierung sofort beendet werden [Re00]. Weiter müssen die Vorgänger- und Nachfolgerbeziehungen entsprechend angepasst werden. Bei der Verwendung der Kantenobjekte muss in der Delta-Schicht durch Aufnahme der neuen Kanten und durch Löschen der alten Kanten der neue Kontrollfluss hergestellt werden. In der anderen Variante müssen dazu die Vorgänger und die Nachfolger des gelöschten Knotens in die Schicht kopiert und deren Liste mit den Nachfolgern bzw. Vorgängern entsprechend abgeändert werden.

Zur Beantwortung der Anfrage "Gib mir alle direkten Nachfolge-Aktivitäten von Aktivität Z!" sieht die Delta-Schicht bei der Implementierung ohne Kanten-Objekte zuerst nach, ob sie ein Aktivitäten-Objekt mit der entsprechenden ID gespeichert hat. Wenn ja, so gibt sie dessen Nachfolgerliste zurück. Ansonsten leitet sie die Anfrage an das referenzierte Prozessvorlagen-Objekt oder im Falle von gestapelten Delta-Schichten an das referenzierte Delta-Schicht-Objekt weiter.

Bei einer Implementierung mit Kanten-Objekten holt sich die Delta-Schicht erst einmal die Menge A aller Kanten-Objekte von dem von ihm geänderten Schema, in denen die Aktivität Z als Quelle verzeichnet ist. Dann entfernt sie aus A alle von ihr als gelöscht markierten Kanten und vereinigt sie mit der Menge der durch die Ad-hoc-Modifikationen neu hinzugekommenen Kanten mit Z als Quelle. Die nun vorliegende Menge B enthält alle von Z ausgehenden Kanten, aus denen die Nachfolger gewonnen werden können. Referenziert das Delta-Schicht-Objekt anstatt eines Vorlagen-Objekts wiederum ein Delta-Schicht-Objekt, dann ist für die Bestimmung der Menge A rekursiv dasselbe Verfahren auf dem referenzierten Delta-Schicht-Objekt auszuführen.

Abbildung 3.7 gibt ein Beispiel: Um die Nachfolger-Aktivität von A1 der zweimalig ad hoc modifizierten Prozessinstanz I1 zu bestimmen, wird eine entsprechende Anfrage an das Delta-Schicht-Objekt II gestellt.

Das Delta-Schicht-Objekt muss zur Beantwortung der Anfrage die Menge B_3 der von A1 ausgehenden Kanten in dem vorliegenden Laufzeitschema bestimmen. Dazu muss es die Menge A_3 der Kanten mit A1 als Quelle von dem von ihm geänderten Schema holen, das durch das Delta-Schicht-Objekt I und durch das Vorlagen-Objekt repräsentiert wird, und dann daraus die von ihm gelöschten Kanten entfernen, sowie die hinzugefügten hinzufügen. Aus Sicht des Delta-Schicht-Objekts I ist die Menge A_3 die Menge B_2 der ausgehenden Kanten von A1 in dem durch die Delta-Schicht I und dem Vorlagen-Objekt repräsentierten Schema. Deshalb ergeht vom Delta-Schicht-Objekt II eine entsprechende Anfrage an das Delta-Schicht-Objekt I. Es beantwortet diese mit der Menge $B_2 = \{(A1 \rightarrow A21), (A1 \rightarrow AY)\}$. Nach dem Entfernen der gelöschten Kante $A1 \rightarrow AY$ und dem Hinzufügen der neuen Kante $A1 \rightarrow AX$ liegt mit

[4] in Tupel-Schreibweise: (A2, A23)

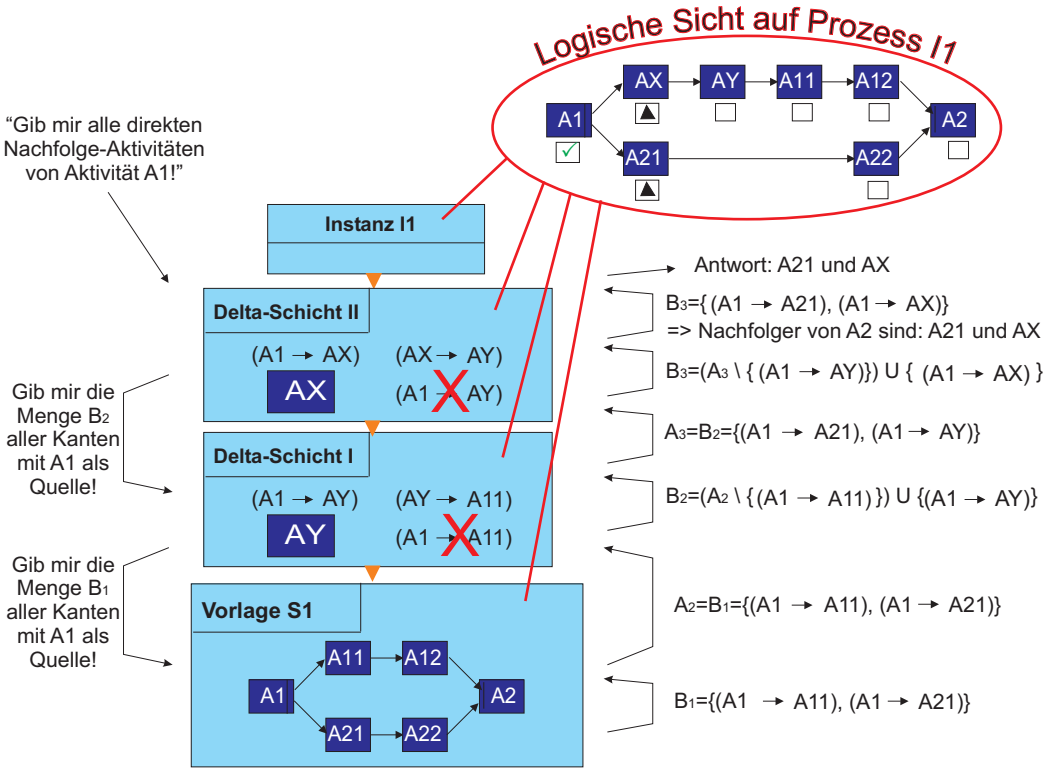

Abbildung 3.7: Nachfolgerbestimmung bei Implementierung mit Kantenobjekten

$B_3 = \{(A1 \rightarrow A21), (A1 \rightarrow AX)\}$ die Menge der von A1 ausgehenden Kanten im abgebildeten Prozess I1 vor, aus der die Nachfolgeraktivitäten A21 und AX von A1 ermittelt werden können.

Für die Bestimmung der Menge B_2 muss das Delta-Schicht-Objekt I analog vorgehen: Zuerst bestimmt es die Menge $A_2 = B_1$ der ausgehenden Kanten von A1 in dem von dieser Delta-Schicht abgeänderten Schema durch eine entsprechende Anfrage an das Vorlagen-Objekt. Das Vorlagen-Objekt liefert dafür die Menge $B_1 = \{(A1 \rightarrow A11), (A1 \rightarrow A21)\}$. Dann entfernt es daraus die gelöschte Kante $A1 \rightarrow A11$ und fügt die Kante $A1 \rightarrow AY$ hinzu. Damit liegt die Menge $B_2 = \{(A1 \rightarrow A21), (A1 \rightarrow AY)\}$ vor, die sie an das Delta-Schicht-Objekt II als Antwort auf dessen Anfrage zurückgibt.

Die Variante mit Kanten-Objekten hat damit den Nachteil, dass stets der Zugriff auf alle Delta-Schicht-Objekte und dem zugrunde liegenden Prozess-Vorlagen-Objekt notwendig ist, um die ein- oder ausgehenden Kanten einer Aktivität zu bestimmen. Bei der Variante ohne Kantenobjekte entfallen die zusätzlichen Zugriffe, wenn die entsprechende Aktivität bereits in der obersten Delta-Schicht vorhanden ist. Dieser Unterschied fällt besonders stark ins Gewicht, wenn mehrere Delta-Schichten übereinander gestapelt werden.

3.6 Migration verzerrter Instanzen

Wie sich eine verzerrte Instanz auf die neue Version S' der zugrunde liegenden Prozessdefinition migrieren lässt, ist abhängig von dem Überlappungsgrad der Schemaänderungen ΔS und den Instanzänderungen ΔS_I (vgl. Abschnitt 2.3.2).

3.6.1 Migration bei disjunkten Änderungsoperationen

Sind die Schema- und die Instanzänderungen disjunkt, d. h. $\Delta S \cap \Delta S_I = \emptyset$, so wäre ein Ansatz für die Migration, die Referenz auf das Prozessvorlagen-Objekt in der Delta-Schicht auf die neue Version „umzubiegen“. Dies kann aber bei der Implementierungsvariante ohne Kanten-Objekte zu Problemen führen (vgl. Abbildung 3.8): A1, A2 und A3 seien drei sequentiell in Folge geschaltete Aktivitäten. Das Einfügen eines Knotens A23 zwischen A2 und A3 auf Instanzebene führt dazu, das eine Kopie A2* von A2 in der Delta-Schicht angelegt wird und darin die ID von A23 anstelle der ID von A3 in die Nachfolger-Liste aufgenommen wird. Durch das Einfügen der Aktivität A12 zwischen A1 und A2 auf Schemaebene wird die ID von A1 durch die ID von A12 in der Vorgängerliste von A2 ersetzt. Die Anfrage “Gib mir die direkten Vorgänger von A2” liefert jedoch weiterhin A1 als Vorgänger, da die Delta-Schicht die Vorgängerliste von A2* zurückgibt, die aber durch die Schemaevolution nicht angepasst wurde und deshalb immer noch die ID von A1 enthält.

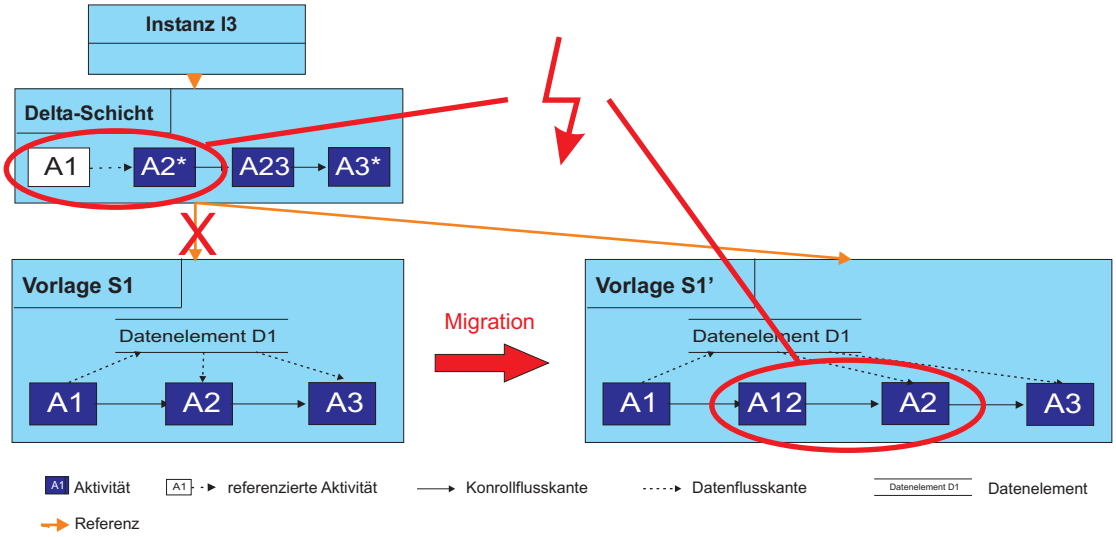

Abbildung 3.8: Inkorrekte Migration verzerrter Instanzen bei Realisierung ohne Kantenobjekte

Eine Lösung dieses Problems besteht darin, auf das Prozessvorlagen-Objekt, das die neue Version des Schemas repräsentiert, ein leeres Delta-Schicht-Objekt aufzusetzen. Darauf werden dann die gleichen dynamischen Änderungen angewendet und vom entsprechenden Instanz-Objekt refe-

renziert. Da bei dieser Vorgehensweise das Einfügen von A12 vor dem Einfügen von A23 erfolgt, weist A2 schon vor dem Kopieren in die Delta-Schicht A12 als seinen Vorgänger aus. Nach dem Kopieren gilt daher das Gleiche für die Kopie A2* von A2. Die Anfrage wird somit korrekt beantwortet. Die Abfolge der auf Instanz- und Typebene vorgenommenen Änderungen zu vertauschen war zulässig, da vorausgesetzt wurde, dass sich die Änderungen nicht überlappen[5].

Bei der Implementierung mit Kantenobjekten tritt das Problem des "vergessenen" Vorgängers nicht auf (siehe Abbildung 3.9): Die Delta-Schicht enthält hier durch das dynamische Einfügen von A23 – abgesehen von dem Aktivitäten-Objekt für A23 – nur die neuen Kanten A2→A23 und A23→A3. A2→A3 wird dagegen als aufgehoben markiert. Auf Schemaebene werden beim

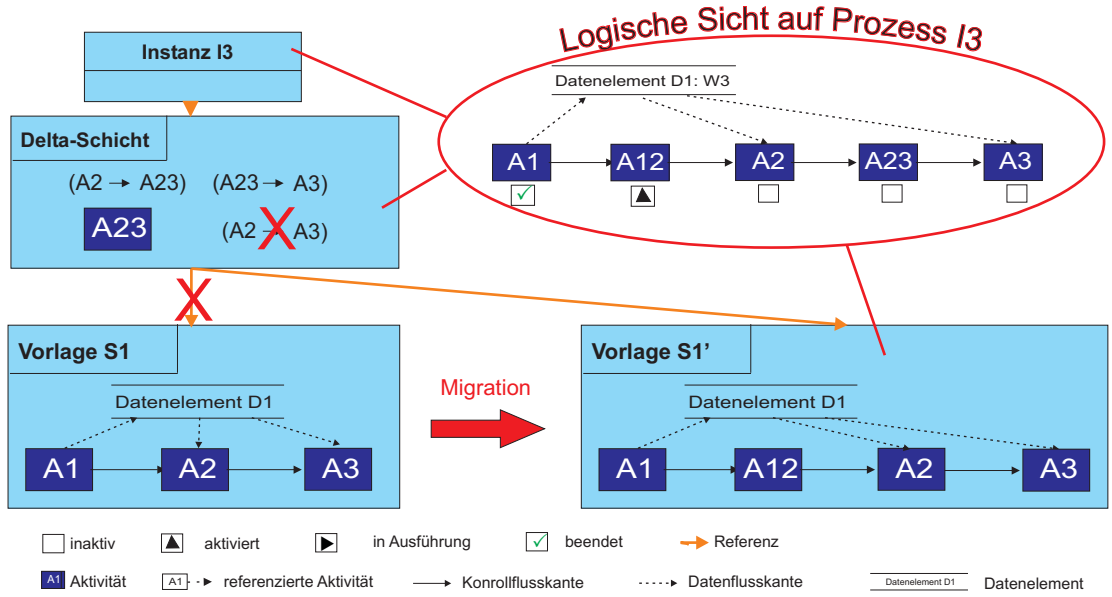

Abbildung 3.9: Problemlose Migration verzerrter Instanzen bei Realisierung mit Kantenobjekten

Einfügen von A12 die Kante A1→A2 komplett entfernt und dafür A1→A12 und A12→A2 eingefügt. Zu Bearbeitung der obigen Anfrage holt sich die Delta-Schicht erst alle Kanten mit A2 als Ziel von der Prozessvorlage. Das Vorlagen-Objekt gibt in diesem Beispiel nur A12→A2 an die Delta-Schicht zurück. Da die Kante in der Delta-Schicht nicht als gelöscht markiert ist und dort keine weiteren Kanten mit A2 als Ziel vermerkt sind, liefert die Delta-Schicht schließlich A12 als die Vorgängeraktivität von A2.

Überlappen sich jedoch Schema- und Instanzänderungen, so müssen andere Migrationsstrategien angewendet werden [Ri04]. Diese werden im folgenden beschrieben.

[5]Disjunkte Änderungen sind kommutativ (vgl. [Ri04])

3.6.2 Migration bei äquivalenten Änderungsoperationen

Wie in Kapitel 2.3.2 angedeutet, dürfen bei einer gegebenen Ausführungsäquivalenz von modifiziertem Instanzschema und neuer Schemaversion – auf logischer Ebene – die Schemaänderungen nicht auf das Laufzeitschema der verzerrten Instanz angewendet werden, da die dynamische Änderung der Instanz die Schemaänderungen bereits vorweggenommen hat. Übertragen auf die vorgestellte Architektur würde das bedeuten, dass keine Änderungen an der internen Repräsentation dieser Instanz vorzunehmen sind.

Dann würde die Instanz aber auch nach der Migration noch indirekt über das Delta-Schicht-Objekt das Vorlagen-Objekt der alten Version S referenzieren, anstatt das der neuen Version S'. Die Instanz besäße demnach (faktisch) den falschen Typ S.

Aufgrund der Ausführungsäquivalenz der gegenüber der alten Version der Schemavorlage verzerrten Instanz und der neuen Schemaversion kann die Instanz als unverzerrt gegenüber der neuen Version betrachtet werden. Für das Laufzeitschema der Instanz gilt nämlich: $S'_I = S_I + \Delta S_I \equiv_{trace} S + \Delta S = S'$. Die richtige Migrationsstrategie für ausführungsäquivalente verzerrte Instanzen ist es deshalb, das Delta-Schicht-Objekt zu verwerfen und direkt das Vorlagen-Objekt, das die neue Version repräsentiert, zu referenzieren. Damit wird zum einen ausgedrückt, dass die Instanz nun von dem neuen Typ S' ist, und zweitens, dass die Instanz gegenüber dem neuen Typ nicht mehr verzerrt ist.

3.6.3 Migration bei subsumptions-äquivalenten Änderungsoperationen

Ist die Instanz-Änderung ΔS_I subsumptions-äquivalent zu den Typ-Änderungen ΔS, d. h. $\Delta S_I \prec \Delta S$, so kann die gleiche Strategie angewendet werden, wie im Falle der Äquivalenz. Mit dem Umhängen auf die neue Version S' werden nämlich implizit die in ΔS_I gegenüber ΔS fehlenden Änderungen $\Delta S \setminus \Delta S_I$ nachgeholt. Die Instanz ist gegenüber der neuen Version S' wiederum unverzerrt.

Gilt jedoch $\Delta S_I \succ \Delta S$, so wird auch das ursprüngliche Delta-Schicht-Objekt der Instanz verworfen und das neue Vorlagen-Objekt referenziert. Da aber ΔS_I mehr Änderungen am Laufzeitschema der Instanz vornimmt als die Schemaänderungen ΔS, ist die Instanz nach der Migration noch gegenüber der neuen Schemaversion S' verzerrt und zwar mit dem Bias $\Delta S'_I = (\Delta S_I \setminus \Delta S)$: $S_I{*'} = S_I + \Delta S + (\Delta S_I \setminus \Delta S) \equiv_{trace} S + \Delta S + (\Delta S_I \setminus \Delta S) = S' + (\Delta S_I \setminus \Delta S) = S' + \Delta S'_I$. Deshalb muss noch zwischen Instanz-Objekt und dem Vorlagen-Objekt der neuen Version S' ein leeres Delta-Schicht-Objekt eingefügt und darauf die Änderungen $\Delta S'_I = (\Delta S_I \setminus \Delta S)$ angewendet werden. $(\Delta S_I \setminus \Delta S)$ muss dynamisch während der dritten Phase des Migrationsprozesses (siehe 2.3.5) aus den beiden Änderungshistorien berechnet werden. [Ri04] stellt effiziente Algorithmen dafür vor.

3.6.4 Migration bei partiell äquivalenten Änderungensoperationen

Bei partiell äquivalenten Änderungsoperationen ist die Instanz nach der Migration ebenfalls gegenüber der neuen Version S' verzerrt. Das neue Delta-Schicht-Objekt muss die Abweichung $\Delta S'_I$ der migrierten Instanz gegenüber der neuen Version der Vorlage S' repräsentieren, so dass gilt: $S_I{*}' = S' + \Delta S'_I$.

Wie in der Theorie zur Migration von verzerrten Instanzen in Abschnitt 2.3.2 angedeutet, kann die Abweichung $\Delta S'_I$ oft nur mithilfe der Benutzer oder anhand von im System hinterlegten Regeln bestimmt werden.

Dynamische Datenfluss-Änderungen einer Instanz werden auf gleiche Art und Weise verwaltet: Die Delta-Schicht speichert die Änderungen gegenüber dem in dem Prozessvorlagen-Objekt vorgegebenen Datenfluss. Zugriffe auf Elemente des Datenflusses erfolgen dann, wie für Kontrollflusselemente skizziert. Die Migration läuft gleich ab.

3.7 Delta-Schicht und andere Prozess-Modelle

Der Delta-Schicht-Ansatz zur Repräsentation der Abweichung eines Instanzschemas gegenüber der Vorlage und der vorgestellte Migrationsablauf können prinzipiell uneingeschränkt auf andere Prozess-Modelle übertragen werden.

Wenn aber wie in ADEPT nur Instanzen migriert werden dürfen, bei denen die Änderungen mit dem bisherigen Prozessablauf vereinbar sind, müssen Informationen sowohl über den momentanen Ausführungsstand als auch über den bisherigen Verlauf vorliegen. Damit ist die Übertragung nur auf Prozess-Modelle sinnvoll, die entsprechende Informationen z. B. in Form einer Ausführungshistorie bereitstellen (siehe z. B. [RRD04]).

Abbildung 3.10 gibt ein Beispiel, wie eine Abweichung des Instanzschemas gegenüber des Vorlagenschemas bei Ein-Marken-Petri-Netzen (siehe [RD02, S. 3-14 ff.]) mithilfe der Delta-Schicht repräsentiert werden kann: Bei der Instanz I1 ist gegenüber der Vorlage S die Transition T3 alternativ zwischen Stelle S1 und S2 eingefügt worden, indem der zu ändernde Teilgraph, hier begrenzt durch S1 und S2, in die Delta-Schicht kopiert und darin die neue Transition T3 eingefügt wurde. Die Frage, welche Transitionen schalten können, wenn die Stelle S1 besetzt ist, wird dann von der Delta-Schicht mit T1 und alternativ T3 beantwortet. Dieselbe Frage beantwortet das Vorlagen-Objekt für Instanz I2 nur mit T1.

3.8 Proof-Of-Concept-Prototype

Für die Konzeptvalidierung haben wir im Rahmen eines Praktikums (siehe [BGL03]) und dieser Diplomarbeit einen Prototypen eines einfachen adaptiven Prozess-Management-Systems, den *Demonstrator*, entwickelt. Er ermöglicht es, Prozessvorlagen anzulegen und darauf basierende

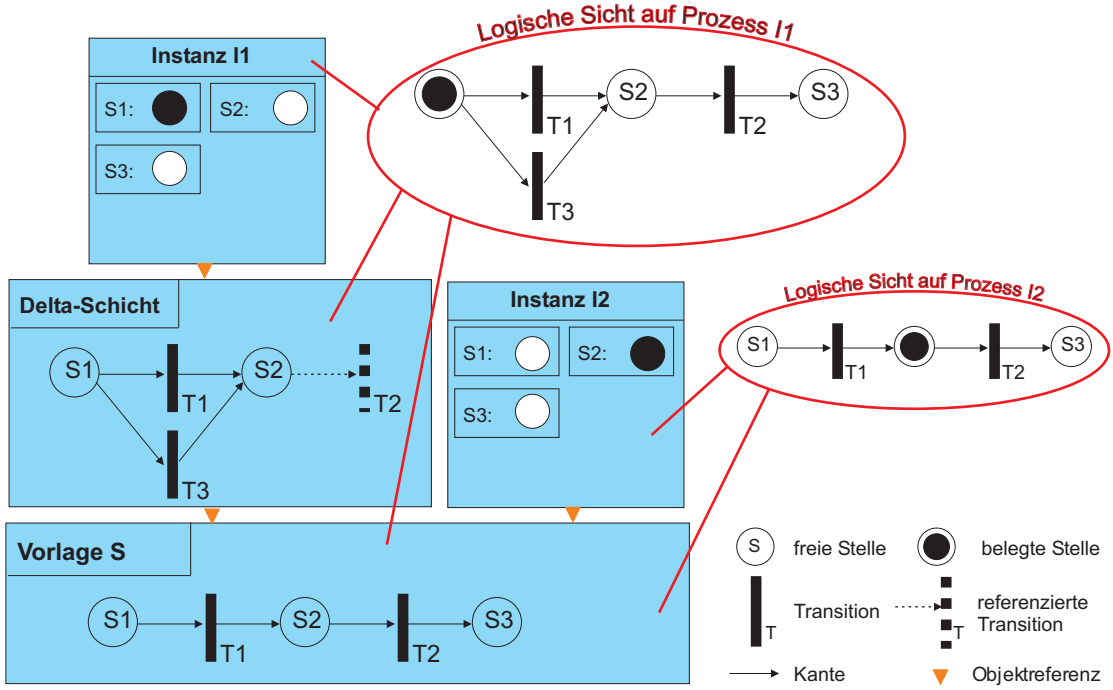

Abbildung 3.10: Die Delta-Schicht bei Petri-Netzen

Instanzen zu starten. Die Instanzen können dann zur Laufzeit dynamisch gegenüber ihrer Vorlage abgeändert werden. Bei der an eine Schemaevolution anschließenden Migration der auf dem abgeänderten Schema basierenden Instanzen werden auch die verzerrten Instanzen berücksichtigt. Sie werden abhängig vom Überlappungsgrad der Instanz- und Schemaänderungen an die neue Version des Schemas angepasst. Vor einer Migration einer Instanz überprüft der Demonstrator, ob die Instanz überhaupt verträglich mit der Schemaänderungen ist. Liegt eine verzerrte Instanz vor, prüft er zusätzlich, ob es durch die Anpassung an die neue Vorlagenversion nicht zu inkorrekten Kontroll- und Datenflüssen kommt. Im Falle von Unverträglichkeit oder von strukturellen Problemen weist er die Instanz von der Migration zurück.

Anhang B demonstriert anhand einiger Screenshots die Fähigkeiten des Demonstrators.

Wir haben in der Proof-Of-Concept-Implementierung die Instanzen entsprechend der Abbildung 3.5 intern abgebildet und den Migrationsvorgang, wie für Implementierungen ohne Kantenobjekte beschrieben, realisiert. Abbildung 3.11 zeigt den entsprechenden Ausschnitt aus dem Klassendiagramm des Prototypen.

Instanzen werden im Prototyp durch ein `Instance`-Objekt repräsentiert. Es enthält keine Schemainformationen, sondern nur Informationen über den Laufzeitzustand der Instanz und indirekt die Datenwerte. Das Laufzeitschema der Instanz wird im unverzerrten Fall durch das referenzier-

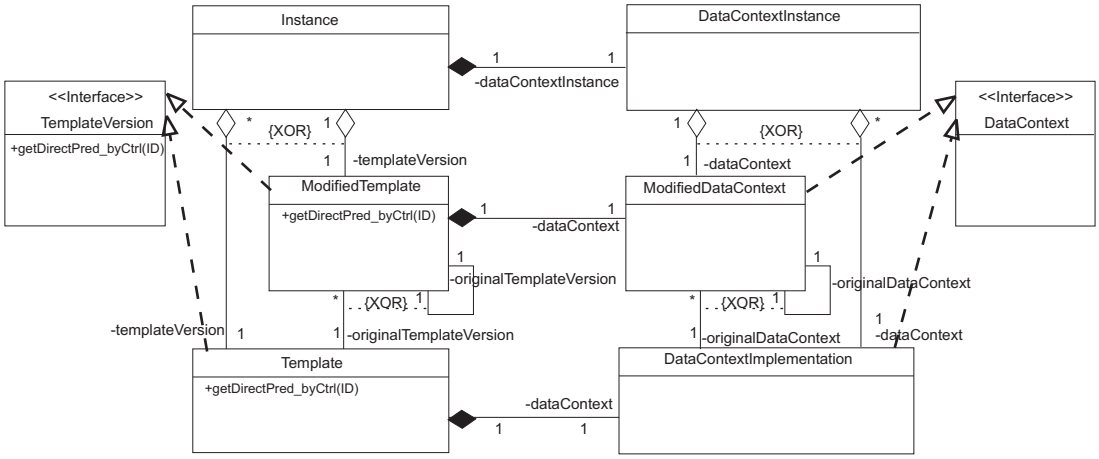

Abbildung 3.11: Repräsentation der Prozessinstanzen und -vorlagen beim Demonstrator

te `Template`-Objekt repräsentiert, im verzerrten Fall durch die Kombination `Template`-Objekt und `ModifiedTemplate`-Objekte.

Das `Template`-Objekt korrespondiert mit dem Vorlagen-Objekt aus diesem Kapitel und bestimmt sowohl den Prozesstyp einer Instanz als auch das Schema, nachdem die Instanzen dieses Prozesstyps im Allgemeinen ablaufen. Wie in der Theorie geschildert, referenzieren alle Instanzen desselben Prozesstyps – direkt oder indirekt – dasselbe `Template`-Objekt. Ein `ModifiedTemplate` entspricht der Delta-Schicht. Es speichert die geänderten Abschnitte des Prozessgraphen. Wie in diesem Kapitel gefordert, weist das `ModifiedTemplate`-Objekt die gleiche Schnittstelle auf wie das Vorlagen-Objekt `Template`: Beide implementieren das Interface `TemplateVersion`. Dadurch wird der Zugriff auf diese beiden Objekte transparent.

Jedes `ModifiedTemplate` referenziert entweder das typ-bestimmende `Template`-Objekt oder wiederum ein `ModifiedTemplate`-Objekt im Falle übereinander gestapelter Delta-Schichten[6]. Kann die Delta-Schicht die an sie gerichtete Frage, wie z. B. „Gib mir alle direkten Vorgängeraktivitäten der Aktivität mit der ID `inID`“, nicht beantworten, delegiert sie die Frage an das referenzierte, die Rolle „`originalTemplateVersion`“ innehabende `Template`- oder `ModifiedTemplate`-Objekt weiter, wie das Code-Beispiel 3.1 demonstriert:

Code 3.1 (Methode `getDirectPred_byCtrl(Integer inID)` von `ModifiedTemplate`)

```
public ActivityList getDirectPred_byCtrl(Integer inID){
```

[6]Das Stapeln mehrerer Delta-Schichten wird momentan im Prototyp noch nicht verwendet. Die Architektur ist dafür aber ausgelegt.

```
  //Are there any information about the activity with ID inID
  //in the delta-layer?

  Activity currActivity = (Activity) activityID2ActivityMap.get(inID);
  if (currActivity!=null){

    //There are information about the activity with ID inID
    //in the delta-layer.
    //So answer the question with this information.

    return currActivity.getCtrlPred(); //answer the question

  }
  else{

   //There are no information about activity with ID inID
   //in the delta-layer, so delegate the question
   //to the original template, that is modified by this delta-layer.

    return originalTemplateVersion.getDirectPred_byCtrl(inID);
  }
}
```

Das Code-Beispiel gibt – aus Übersichtsgründen in etwas vereinfachter Form – die Methode `getDirectPred_byCtrl(Integer inID)` von `ModifiedTemplate` wieder, die als Antwort eine Liste mit den IDs aller Aktivitäten zurückliefert, welche im Kontrollfluss direkte Vorgänger der Aktivität mit der ID `inID` sind.

Zuerst wird mit dem Aufruf `activityID2ActivityMap.get(inID)` überprüft, ob die betroffene Aktivität im Rahmen einer Ad-hoc-Modifikation für eine Anpassung in die Delta-Schicht kopiert wurde. Ist dies der Fall, so ist der Rückgabewert eine Referenz auf das zur Aktivität mit der ID `inID` gehörige Aktivitäten-Objekt. Mit dem Aufruf der Methode `getCtrlPred()` des entsprechenden `Activity`-Objekts bekommt man dann die aktuell gültige Liste mit den IDs der Vorgängeraktivitäten. Wurde die Aktivität im Rahmen der Ad-hoc-Modifikation jedoch nicht geändert, so liefert der Methodenaufruf `activityID2ActivityMap.get(inID)` eine `null`-Referenz zurück. In diesem Fall kann nur das durch diese Delta-Schicht abgeänderte Schema die Antwort geben. Somit wird die Anfrage delegiert.

Der Datenfluss wird analog zum Kontrollfluss gehandhabt. Das zum `Template`-Objekt gehörende `DataContextImplemention`-Objekt dient als Vorlage für den Datenfluss einer Instanz des durch das `Template`-Objekt festgelegten Prozesstyps. Der Datenfluss gibt vor, welche Daten von welchen Aktivitäten geschrieben und von welchen Aktivitäten wieder gelesen werden. Das `ModifiedDataContext`-Objekt repräsentiert analog zum ModifiedTemplate für eine Instanz die im Rahmen einer dynamischen Änderung entstandenen Abweichungen des Datenflusses von der

Vorlage. Bei einer verzerrten Instanz erfolgen Zugriffe auf den Datenfluss dann zuerst über das `ModifiedDataContext`-Objekt. Analog zum `ModifiedTemplate` delegiert es Anfragen zur Beantwortung, wenn die Anfragen Teile des Datenflusses betreffen, die nicht geändert wurden. Das zum `Instance`-Objekt gehörende `DataContextInstance`-Objekt schließlich speichert die instanzspezifischen, von Aktivitäten geschriebenen Datenwerte.

3.9 Zusammenfassung und Ausblick

Mit dem skizzierten Ansatz liegt eine übersichtliche interne Repräsentation von Prozessvorlagen und Instanzen vor, mit der sich die Migration von unveränderten und sogar dynamisch abgeänderten Instanzen einfach realisieren lässt. Der Migrationsvorgang ist schnell und effizient, da er bei diesem Ansatz meistens nur aus dem Umhängen von Referenzen besteht. Außerdem ist der Speicherbedarf gegenüber anderen Ansätzen stark reduziert, da auf Wiederverwendung gesetzt wird – dasselbe Prozess-Vorlagen-Objekt wird z. B. von mehreren Instanzen referenziert — und da die Delta-Schicht nur abgeänderte Abschnitte des Prozessgraphen speichert. Aufgrund dieser Eigenschaften ist dieser Architekturansatz bestens geeignet für einen Einsatz in Hochleistungs-Prozess-Management-Systemen.

Um weitere Anforderungen zu erfüllen, die ein Einsatz in Produktivsystemen mit sich bringt, muss die Architektur noch erweitert werden. Es muss unter anderem ein darauf abgestimmtes Sperren-System entwickelt werden, um den nebenläufigen Zugriff sowohl auf Prozessvorlagen als auch auf Instanzen zu koordinieren bzw. zu synchronisieren.

Auch muss untersucht werden, wie eine Migration durchgeführt wird, wenn eine Instanz partitioniert auf mehreren Servern oder sogar in mehreren Prozess-Management-Systemen abläuft. Diese Situation kann auftreten, wenn zwei parallel angeordnete Prozessabschnitte an verschiedenen Firmenstandorten ausgeführt werden und es z. B. aufgrund einer schlechten Netzwerkanbindung untereinander unmöglich ist, den Prozess von einem PMS alleine kontrollieren zu lassen. Wie sich eine Ad-hoc-Modifikation effizient unter diesen Bedingungen durchführen lässt, wird z. B. in [BRD01] behandelt.

Weiterhin bleibt zu untersuchen, ob durch kleine Änderungen oder Erweiterungen der Architektur in Bezug auf Instanzen der Migrationsprozess im Gesamten optimiert werden kann. Das nächste Kapitel geht dieser Frage nach.

Kapitel 4

Optimierungsansätze

Bei dem in Kapitel 2.3.5 vorgestellten Migrationsablauf durchläuft jede einzelne Instanz alle fünf bzw. sechs Phasen, je nachdem ob die Instanz unverzerrt oder verzerrt gegenüber der Prozessvorlage ist. Der Migrationsprozess kann aber noch optimiert werden. Dieses Kapitel stellt dazu zwei mögliche Ansätze vor:

1. Instanz-Gruppierung nach Laufzeitzustand (Abschnitt 4.1)
2. Meilenstein-Ansatz (Abschnitt 4.2)

Beide Ansätze versuchen, für Instanzen Teile des Migrationsablaufs einzusparen und damit Rechenzeit zu gewinnen.

4.1 Instanz-Gruppierung nach Laufzeitzustand

Existieren in der Menge der zu migrierenden Instanzen einige Instanzen, die sich weder im Laufzeitschema noch im Ausführungszustand unterscheiden, so muss die Verträglichkeitsprüfung nur für eine dieser Instanzen ausgeführt werden. Das Ergebnis kann dann uneingeschränkt auf die anderen Instanzen übertragen werden. Liefert der Test, dass die überprüfte Instanz verträglich ist, so sind auch alle anderen bezüglich Laufzeitschema und Ausführungszustand äquivalenten Instanzen verträglich. Gleiches gilt bei Unverträglichkeit. Es können somit unter Umständen etliche Verträglichkeitsprüfungen eingespart werden:

Gegeben sei ein Prozess S_{sequ}, bei dem n=50 Aktivitäten sequentiell hintereinander angeordnet sind. Nimmt man an, dass nur die Zustände “INAKTIV”, “AKTIVIERT”, “IN AUSFÜHRUNG” und “BEENDET” existieren, so kommt es bei diesem Prozess höchstens zu $\#(S_{sequ}, 50) = 101$ verschiedenen Laufzeitzuständen[1]. Liegen ausschließlich unverzerrte Instanzen vor, so müssen

[1] Zur Berechnung von $\#(S_{sequ}, 50)$ siehe Anhang A

bei Anwendung des Gruppierungsansatzes nur maximal 101 Verträglichkeitsprüfungen bei einer Migration durchgeführt werden, egal wie viele Instanzen dieses Prozesstyps existieren. Befinden sich momentan z. B. 10100 Instanzen des Prozesses in Ausführung, so reduziert sich der Aufwand für die Verträglichkeitsprüfung um den Faktor 100 gegenüber der Variante, bei der alle Instanzen einzeln auf Verträglichkeit geprüft werden.

Die Aussage, dass die Verträglichkeitsprüfung nur für eine Instanz der Gruppe durchgeführt werden muss und dass sich dann das Ergebnis auf die anderen Instanzen der Gruppe übertragen lässt, gilt nicht generell, wenn eine oder mehrere Sync-Kanten im Rahmen der Migration einzufügen sind. Dann kann eine Gruppe nämlich aus unverträglichen und verträglichen Instanzen bestehen, da die Verträglichkeit einer Sync-Kanten-Einfüge-Operation nicht immer nur vom Zustand abhängt, sondern in manchen Fällen auch davon, wann die Quell- und Zielaktivitäten der neuen Sync-Kanten gestartet bzw. beendet wurden: Eine Sync-Kanten-Einfüge-Operation kann auf eine Instanz angewendet werden, wenn die Zielaktivität entweder noch nicht gestartet wurde, die Quellaktivität in einem abgewählten Zweig eines XOR-Blocks liegt und damit den Zustand „ABGEWÄHLT“ besitzt, oder wenn der Startzeitpunkt der Zielaktivität später liegt als der Endzeitpunkt der Quellaktivität [RRD02].

Wird z. B. nur die Sync-Kante $X \rightarrow Y$ eingefügt, dann müssen alle Instanzen gesondert auf Verträglichkeit geprüft werden, die einen der Zustände aufweisen, bei denen die Quellaktivität X bereits beendet ist und die Zielaktivität Y sich entweder in Ausführung befindet oder ebenfalls bereits beendet ist. Die Instanzen, bei denen X nach dem Ende von Y gestartet wurde, sind verträglich und können migriert werden, alle anderen sind unverträglich und müssen von der Migration ausgeschlossen werden. Die Informationen über die Start- und Endzeitpunkte von Aktivitäten sind in der Ausführungshistorie hinterlegt.

Ähnlich verhält es sich, wenn Datenelementen gelöscht werden. Für die Verträglichkeitsprüfung muss u. U. ein Zugriff auf die Datenhistorie einer Instanz erfolgen [RRD02].

Aber nicht nur durch die Reduzierung der aufwändigen Verträglichkeitsprüfungen lassen sich theoretisch Optimierungen erzielen, sondern auch durch das gemeinsame Ändern der Markierungszustände für diese Instanzen: Unterscheiden sich die Instanzen weder im Laufzeitschema noch im Ausführungszustand, so muss nur einmal die Menge der Aktivitäten bestimmt werden, deren Zustände angepasst werden müssen. Bei den anderen Instanzen können diese Informationen wieder verwendet werden, es müssen nur die bereits ermittelten Anpassungen durchgeführt werden. Durch entsprechende Implementierungen ist es sogar möglich, für alle Instanzen die Markierungen zugleich anzupassen (siehe unten).

Um diese Optimierungsstrategien anwenden zu können, muss die in Kapitel 3 vorgestellte Architektur erweitert werden. Sie muss dahingehend modifiziert werden, dass Instanzen nach ihrem Laufzeitzustand gruppiert werden können. Um der Forderung nach gleichem Laufzeitschema nachzukommen, beschränken wir die Betrachtungen auf unverzerrte Instanzen und behandeln verzerrte Instanzen wie bisher geschildert. Die Forderung nach gleichen Laufzeitschemata ist bei allen gegenüber der Vorlage unverzerrten Instanzen erfüllt, da deren Laufzeitschemata äquivalent zu der Vorlage sind.

Da alle Instanzen einer Gruppe denselben Laufzeitzustand aufweisen, ist es vom Gesichtspunkt des Speicherbedarfs her sinnvoll, die Informationen über den Zustand der einzelnen Aktivitäten nicht in jeder Instanz redundant abzuspeichern, sondern nur einmal pro Gruppe innerhalb eines Zustandobjektes. Eine Instanz befindet sich dann in einem bestimmten Zustand, wenn sie das entsprechende Zustandsobjekt referenziert. Alle Instanzen, die das gleiche Zustandsobjekt referenzieren, bilden zusammen eine Gruppe. Abbildung 4.1 demonstriert diesen Sachverhalt.

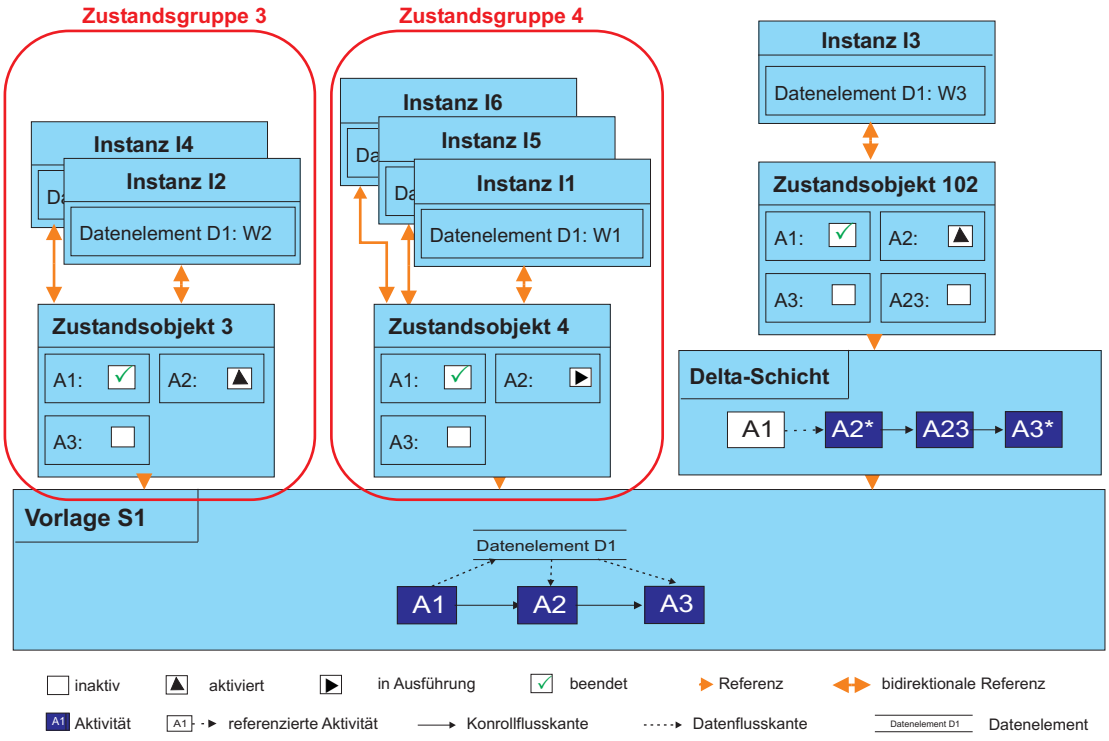

Abbildung 4.1: Architektur mit Zustandsgruppen vor der Migration

Aus Gründen der Wiederverwendung ist es sinnvoll, für die Repräsentation einer dynamisch geänderten Instanz statt des aus Kapitel 3 bekannten Instanz-Objekts, das zusätzlich den Zustand speichert, die hier vorgeschlagene Kombination aus Instanz-Objekt und Zustandsobjekt zu verwenden. Das Zustandsobjekt besitzt dann aber nur die Referenz auf das entsprechende Instanz-Objekt.

Das Zustandsobjekt ermöglicht es auch, dass alle referenzierten Instanzen bei Verträglichkeit auf einmal auf das neue Schema umgehängt werden können: Wie Abbildung 4.1 zeigt, referenzieren nicht die Instanz-Objekte das zugrundeliegende Vorlagen-Objekt, sondern nur die jeweiligen Zustandsobjekte. In der vierten Phase des Migrationsprozesses müssen dann nur die Zustandsobjekte auf das neue Vorlagen-Objekt umgehängt werden, die einen mit der Schemaänderung verträglichen Zustand repräsentieren. Damit werden alle Instanzen der Gruppe gleichzeitig von

der Version $S1$ auf die neue Version $S1'$ der Vorlage migriert. Gleiches gilt für die Instanzanpassung und die Markierungsanpassung: Mit dem Anpassen des entsprechenden Zustandsobjekts werden – logisch gesehen – alle Instanzen dieser Gruppe auf einmal angepasst.

Abbildung 4.2 zeigt die Repräsentation der Instanzen direkt nach Abschluss des Migrationsprozesses.

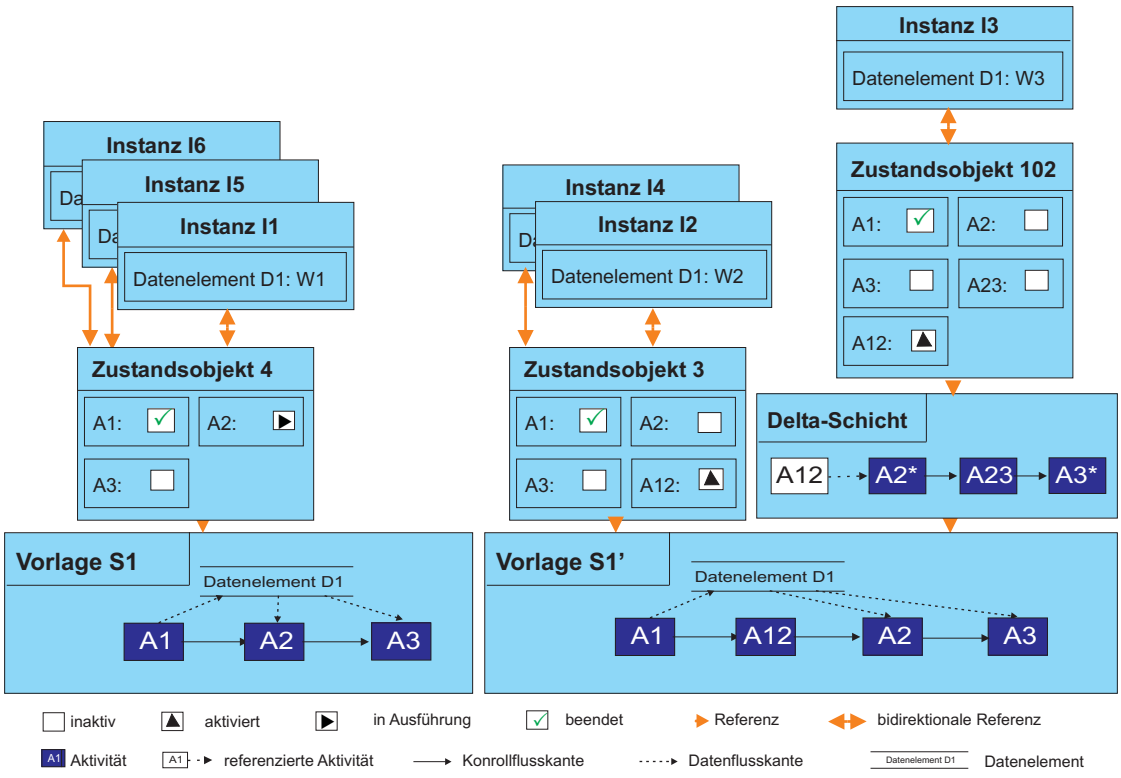

Abbildung 4.2: Architektur mit Zustandsgruppen nach einer Migration

Bei dem oben genannten Beispiel mit einem sequentiellen Prozessgraph müssen dann z. B. nach dem Einfügen einer neuen Aktivität nicht nur maximal 101 Verträglichkeitsprüfungen statt 10100 durchgeführt werden, sondern auch nur höchstens 101 Referenzen statt maximal 10100 umgehängt werden und nur maximal 101 Zustände angepasst werden. Ebenso reduziert sich der Speicherverbrauch für die Zustandsdaten um das 100-fache, da nicht mehr in allen 10100 Instanz-Objekten der Zustand einzeln gespeichert wird sondern nur in maximal 101 Zustandsobjekten.

Gerade in Prozessen mit parallelen Ausführungspfaden kommt es aber schnell zur einer Zustandsexplosion. Das linke Diagramm aus Abbildung 4.3 verdeutlicht dies. Es trägt die Anzahl #(S, 1, x, y, 1) der möglichen Zustände bei einem Prozess S mit zwei parallel angeordneten Pfaden gegenüber der Anzahl x und y der Aktivitäten in den beiden Pfaden ab[2].

[2]Zur Berechnung von $\#(S, 1, x, y, 1)$ siehe Anhang A

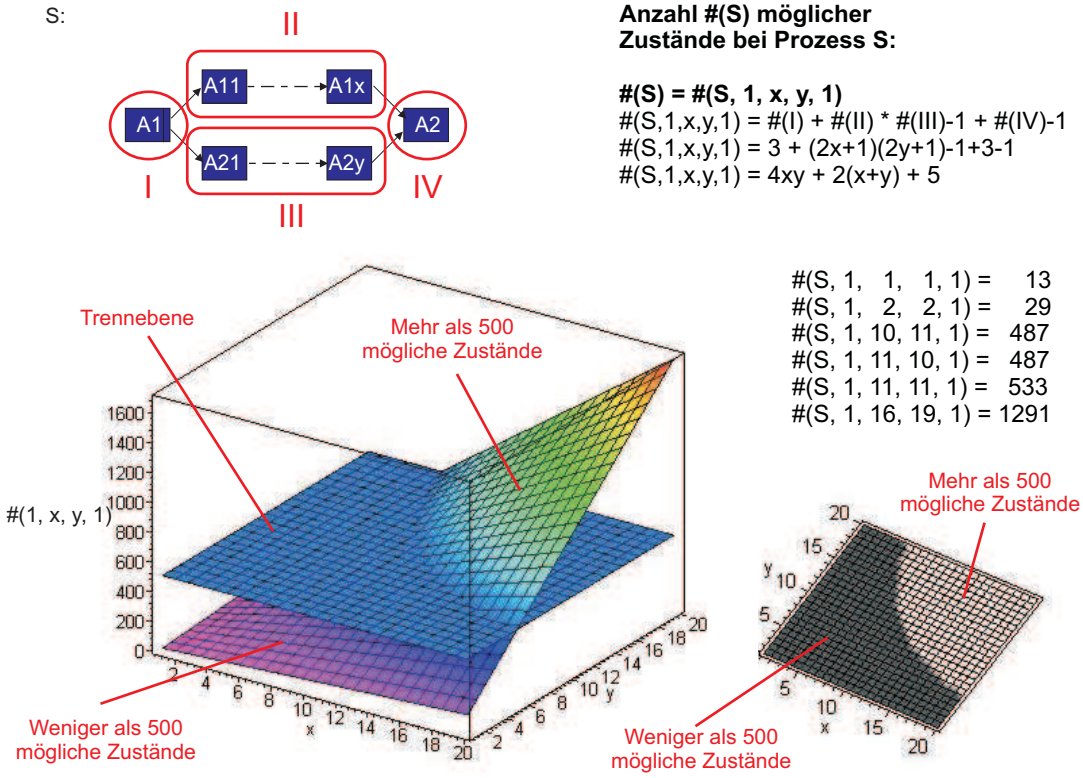

Abbildung 4.3: Anzahl #(S, 1, x, y, 1) möglicher Zustände bei einem Prozess mit zwei parallel angeordneten Pfaden in Abhängigkeit von der Anzahl x und y der Aktivitäten in den Pfaden

Die Folge ist, dass bei Prozessen mit parallelen Ausführungspfaden schon bei geringer Aktivitätenanzahl die Zahl der möglichen Zustände die Zahl der durchschnittlich vorhanden Prozessinstanzen übertreffen kann und damit mehr Zustandsobjekte existieren können als Instanzen, wenn für jeden möglichen Zustand ein Zustandsobjekt angelegt wird. Das rechte Diagramm aus Abbbildung 4.3 demonstriert dies. Es gibt für jede Kombination (x,y) an, ob damit S mehr (helle Fläche) oder weniger (dunkle Fläche) als 500 Zustände aufweist und deshalb die angenommene Zahl der durchschnittlich vorhandenen Instanzen von 500 überschreitet oder nicht.

Bei Prozessen der Form S ergeben sich schon mit 11 Aktivitäten pro parallelem Pfad mehr als 500 mögliche Zustände: $\#(S, 1, 11, 11, 1) = 533$. Gibt es, wie angenommen, 500 Instanzen von diesem Prozess, so sind mindestens 33 Zustandsobjekte ohne zugeordnete Instanzen und damit unnötig angelegt, wenn für alle 533 möglichen Zustände ein Zustandsobjekt angelegt wurde. Diese Zahl erhöht sich rapide mit der Zahl der Aktivitäten: Beinhaltet z. B. der Pfad, der als Bereich II ausgewiesen ist, jetzt 16 seriell angeordnete Aktivitäten und der parallel dazu angeordnete Pfad 19, so berechnet sich die Anzahl $\#(S, 1, 16, 19, 1)$[3] der möglichen Zustände zu 1291. Damit hat

[3] zu den Parametern: relevantes Schema S, eine Aktivität in Bereich I, 16 in II, 19 in III und eine in IV

sich die Zahl der unnötig angelegten Zustandsobjekte auf 791 erhöht, obwohl S nun gerade mal 13 Aktivitäten mehr aufweist.

Um nicht unnötig viel Speicherplatz zu verschwenden, sollten deshalb zu einem Zeitpunkt nur Zustandsobjekte existieren, denen auch Instanzen zugeordnet sind.

Beim Weiterschalten der Instanz muss dann für einige Zustände erst das entsprechende Zustandsobjekt generiert werden. Cachen von Zustandsobjekten, die besonders häufig angenommene Zustände repräsentieren, kann den Aufwand dafür verringern.

Ein Weiterschalten der Instanz erfordert stets mehrere Schritte: Zuerst muss die Instanz aus der entsprechenden Gruppe entfernt werden. Danach muss das nun relevante Zustandsobjekt gefunden und referenziert werden. Für das schnelle Auffinden des neu zu referenzierenden Zustandsobjektes bietet sich eine Map an. In einer Map wird zu einem Schlüssel der zugehörige Wert gespeichert. Als Schlüssel dient eine Codierung für den Zustand. Über diese Codierung kann man dann mithilfe der Map auf das entsprechende Zustandsobjekt zugreifen. Beim Weiterschalten der Instanz wird somit zuerst die Codierung des neuen Zustandes berechnet, dann wird damit das neue Zustandsobjekt herausgesucht. Sollte das Zustandsobjekt nicht existent sein, so muss es zu diesem Zeitpunkt erzeugt werden. Schließlich wird die Instanz auf das neue Zustandsobjekt umgehängt. Die Codierung des neuen Zustandes sollte einfach aus der Codierung des alten Zustandes berechnet werden können. Ein Codierung muss nicht unbedingt eindeutig zu einem Zustand gehören, d. h. die Abbildung Codierung – Zustand muss nicht bijektiv sein. Bei einer nicht bijektiven Abbildung können dann auf eine Codierung mehrere Zustandsobjekte fallen. Beim Weiterschalten muss dann noch zusätzlich das richtige Zustandsobjekt herausgesucht werden.

Wird die Aktivität A2 der Instanz I4 aus Abbildung 4.4 gestartet, so wird zuerst die bidirektionale Referenz vom Zustandsobjekt 3, das den alten Zustand repräsentiert, auf das Instanz-Objekt I4 aufgelöst. Dann wird die Codierung für den neuen Zustand, bei dem A2 als “IN AUSFÜHRUNG” markiert ist, berechnet. In diesem Fall lautet diese Zustandscodierung 4. Damit wird nun schnell über die Map auf das richtige Zustandsobjekt zugegriffen, um dort eine wiederum bidirektionale Referenz auf das Instanz-Objekt I4 zu setzen. I4 befindet sich damit in dem neuen Zustand, der durch das Zustandobjekt 4 repräsentiert wird.

Zusammenfassend kann man sagen, dass das Umschalten relativ aufwändig ist. Treffen zu einem Zeitpunkt sehr viele Instanzweiterschaltungen aufeinander, so wirkt sich das negativ auf die Performance aus.

Hinzu kommt, dass das parallele Umschalten von (vielen) Instanzen zu parallelen Schreib- und Lesezugriffen auf die vorgestelle Architektur führt (z. B. Austragen von mehreren der bijektiven Instanzreferenzen aus demselben Zustandsobjekt). Zur Vermeidung von Inkonsistenzen müssen Sperren ein isoliertes Schreiben garantieren. Das Warten auf die Freigabe von Sperren kostet wiederum wertvolle Performance.

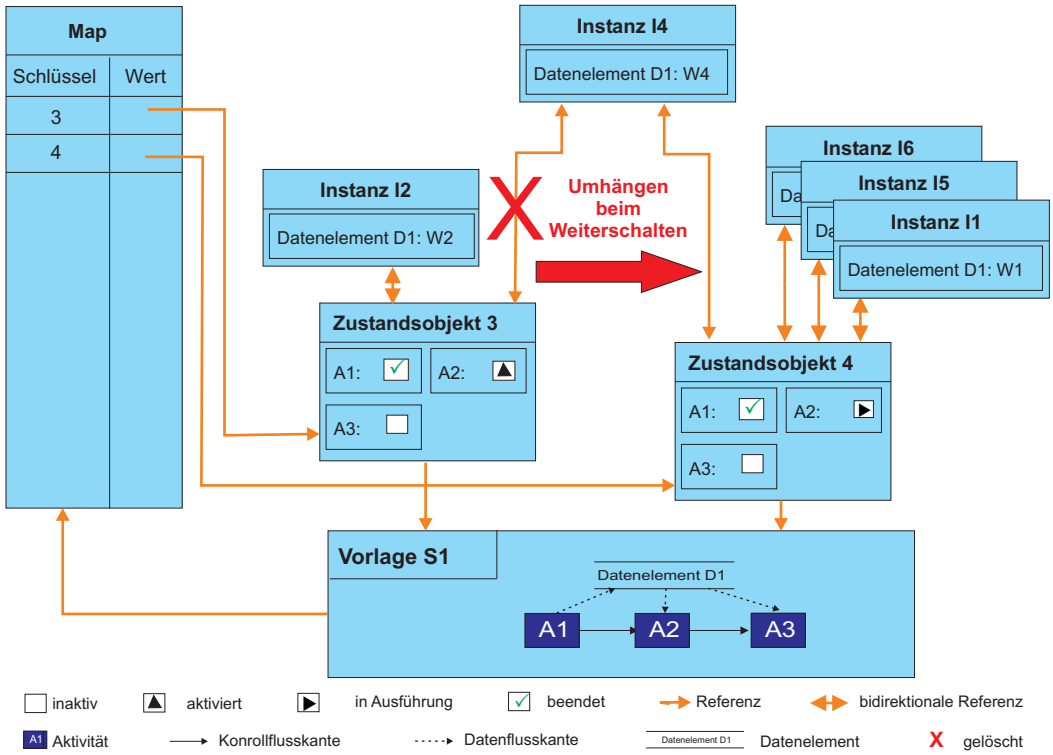

Abbildung 4.4: Weiterschalten einer Instanz bei einer Architektur mit Zustandsgruppen

4.2 Meilenstein-Ansatz

Ein alternativer Ansatz, Verträglichkeitsprüfungen einzusparen, ist der *Meilenstein-Ansatz*. Dabei wird der ganze Prozessgraph in mehrere Abschnitte eingeteilt, wie z. B. in Abbildung 4.5 gezeigt. Gemerkt wird dann, in welchen Abschnitt eine Instanz bereits vorgerückt ist. Dazu wird für jeden Abschnitt eine Liste der Instanzen gespeichert, die den entsprechenden Abschnitt erreicht, aber noch nicht überschritten haben. Beim Überschreiten einer Abschnittsgrenze, dem Meilenstein, wird die Instanz in die Liste des neuen Abschnittes umgehängt. Überschritten ist ein Meilenstein, sobald eine Aktivität aus dem von diesem und dem nächsten Meilenstein begrenzten Prozessabschnitt ausgeführt wird.

Verträglichkeitsprüfungen können deshalb eingespart werden, da alle Instanzen, die noch nicht in den ersten von einer Änderung betroffenen Bereich vorgerückt sind, automatisch als verträglich angenommen werden können. Wurden alle Meilensteine an Stellen im Graph gesetzt, an denen jeweils nur ein Ausführungspfad und nicht mehrere, parallel oder alternativ angeordnete Pfade existieren, so kann weiterhin gesagt werden, dass alle Instanzen, die den ersten von einer Änderung betroffenen Abschnitt überschritten haben, nicht mehr migrierbar sind, da sie zu weit

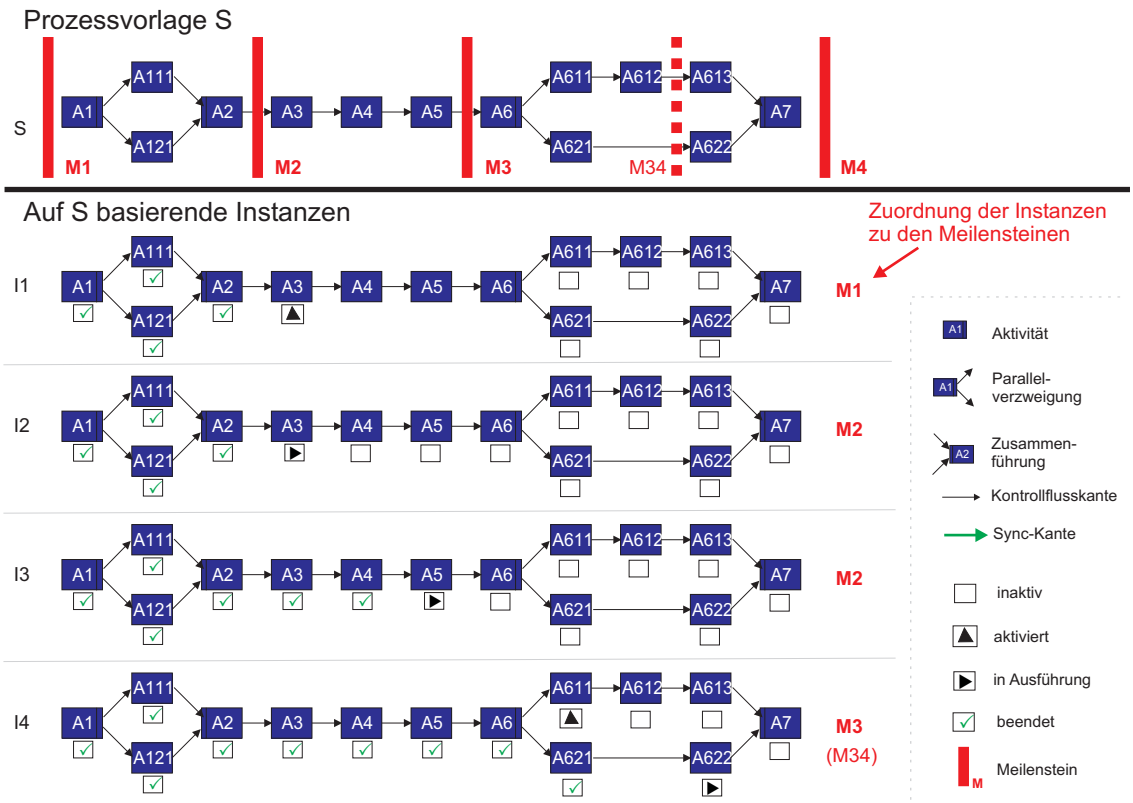

Abbildung 4.5: Architektur mit Meilensteinen

fortgeschritten sind. Es müssen dann nur noch die Instanzen innerhalb des besagten Abschnittes auf Verträglichkeit geprüft werden.

Wird z. B. in den Graph S aus Abbildung 4.5 zwischen die Aktivitäten A3 und A4 eine weitere Aktivität eingefügt, so sind alle von M1 referenzierten Instanzen, wie z. B. I1, verträglich, alle ab inklusive M3 referenzierten dagegen unverträglich, wie z. B. die Instanz I4. Alle Instanzen, die M2 aber nicht M3 überschritten haben, müssen einzeln auf Verträglichkeit geprüft werden: Instanzen, bei denen A4 maximal aktiviert ist, sind verträglich, Instanzen bei denen sich A4 bereits in Ausführung befindet oder bei denen A4 schon beendet wurde, sind nicht verträglich. Somit ist die Instanz I2 verträglich und I3 unverträglich, obwohl beide demselben Meilenstein M2 zugeordnet sind.

Ohne die Bedingung an die Position der Meilensteine müssen alle Instanzen, die nicht von vorne herein als verträglich eingestuft werden konnten, der Verträglichkeitsprüfung unterzogen werden, auch wenn sie bereits den ersten abgeänderten Prozessabschnitt überschritten haben:

Wenn in dem Graph S aus Abbildung 4.5 noch zusätzlich der gestrichelt eingezeichnete Meilenstein M34 berücksichtigt wird, dann ist die Instanz I4 diesem Meilenstein anstatt dem Meilen-

stein M3 zugeordnet, weil sich bei ihr die Aktivität A622 bereits in der Ausführung befindet und die Instanz damit den Bereich zwischen M34 und M4 betreten hat. Wird nun in den dargestellten Prozessgraphen S eine Aktivität zwischen die Aktivitäten A611 und A612 eingefügt, so ist I4 trotzdem noch vom Zustand her mit der Änderung verträglich, obwohl sie bereits den auf den geänderten Bereich folgenden Abschnitt des Prozessgraphen erreicht hat. Die Ausführung in dem Pfad, in dem diese Änderung ausschließlich stattfindet, ist noch nicht zu weit fortgeschritten, um die Instanz unverträglich mit der Änderung zu machen. Deshalb gilt ohne die Bedingung an die Position der Meilensteine nicht, dass alle Instanzen nach dem ersten geänderten Bereich nicht mehr verträglich sind. Aufgrund dessen müssen dabei auch die weiter fortgeschrittenen Instanzen noch auf Verträglichkeit geprüft werden.

Beim Hinzufügen von Sync-Kanten müssen jedoch auch bei der Version mit der Einschränkung bezüglich der Positionierung der Meilensteine noch die Instanzen auf Verträglichkeit überprüft werden, die bereits den ersten abgeänderten Abschnitt überschritten haben. Ist z. B. die neue Sync-Kante die einzigste Modifikation in diesem Abschnitt, dann kann auch eine Instanz, die diesen Bereich bereits überschritten hat, noch verträglich sein, da eine Sync-Kante auch noch zwischen zwei Aktivitäten X und Y eingefügt werden kann, wenn sowohl X als auch Y schon beendet wurden. X muss nur vor dem Start von Y beendet worden sein.

Werden innerhalb einer Änderungstransaktion Datenelemente gelöscht, so müssen hierbei bei beiden Alternativen des Meilenstein-Ansatzes sämtliche Instanzen einzeln auf Verträglichkeit überprüft werden, da für eine Entscheidung über die Verträglichkeit nicht nur der Zustand der Instanz ausschlaggebend ist, sondern auch, ob überhaupt auf das Datenelement lesend oder schreibend zugegriffen wurde (vgl. [RRD02]). Entsprechende Informationen müssen aus den Datenhistorien der Instanzen gewonnen werden.

Der Vorteil gegenüber der in Abschnitt 4.1 genannten Methode der Zustandsgruppierung zur Einsparung von Verträglichkeitsprüfungen ist es, dass nicht bei jedem Weiterschalten von Instanzen aufwändige Datenstrukturanpassungen notwendig sind, sondern nur, wenn Meilensteine überschritten werden.

Trotzdem ist auch in dem Fall, dass kein Meilenstein überschritten wurde, das Weiterschalten gegenüber einem Weiterschalten in einer Umgebung verzögert, bei der keine der hier vorgestellten Optimierungsansätze für den Migrationsprozess zur Anwendung kommt, da bei jedem Weiterschalten überprüft werden muss, ob ein Meilenstein überschritten wurde.

Von Nachteil ist, dass es bei diesem Ansatz nicht mehr möglich ist, erstens mehrere Instanzen auf einmal auf eine neue Version der Prozessvorlage umzuhängen, zweitens die Instanzanpassung für mehrere Instanzen auf einmal vorzunehmen und drittens die nach einer Migration erforderlichen Markierungsanpassungen auf einmal für mehrere Instanzen durchzuführen.

4.3 Zusammenfassung und Alternativen

Vergleicht man die beiden Varianten miteinander, dann kann gesagt werden, dass das Weiterschalten einer Instanz bei der Meilenstein-Methode gegenüber der zuvor vorgestellen Optimierungsvariante mit der Gruppierung der Instanzen nach dem Laufzeitzustand beschleunigt ist. Dafür müssen allerdings bei der Meilenstein-Methode auch Geschwindigkeitseinbußen bei der Migration gegenüber der anderen Variante in Kauf genommen werden.

Generell muss abgewägt werden, ob eine beschleunigte Migration ein verzögertes Weiterschalten von Instanzen überhaupt rechtfertigt und ob damit die Optimierungsstrategien überhaupt angewendet werden sollen. Das Weiterschalten von Instanzen zählt zu den am häufigsten aufzutretenden Ereignissen bei Prozess-Management-Systemen. In großen PMS-Systemen müssen eventuell zu einem Zeitpunkt tausende von Aktivitäten weitergeschaltet werden. Es gehört deshalb besonders effizient implementiert. Die Migration von Instanzen kommt verglichen dazu relativ selten vor.

Eine sinnvolle Alternative ist es, die Architektur aus Kapitel 3 beizubehalten und die Migration auf konventionelle Weise durchzuführen, aber die Migration von Instanzen, auf die momentan nicht zugegriffen wird, in Tagesbereiche, z. B. Nacht, zu verlegen, in denen verhältnismäßig wenig anderweitige Zugriffe auf das Prozess-Management-System stattfinden. Instanzen, bei denen ein Zustandwechsel anliegt, müssen im Falle der Verträglichkeit allerdings sofort auf das neue Schema angepasst werden, damit sie gleich nach den neuen Vorgaben ablaufen können.

Eine verzögerte Migration (engl.: *lazy migration*) ist auch aus Sicht der Speicherverwaltung sinnvoll. Für die Migration muss die Instanz im Hauptspeicher vorliegen. Wurde auf eine Instanz schon lange nicht mehr zugegriffen, so ist die Wahrscheinlichkeit hoch, dass sie im Rahmen der Speicherverwaltung auf den Hintergrundspeicher ausgelagert wurde. In diesem Fall muss sie für die Migration zuerst wieder in den Hauptspeicher geladen werden. Dafür muss aber erst Platz geschaffen werden. Wird dafür eine Instanz aus dem Arbeitsspeicher verdrängt, auf die häufig zugegeriffen wird, ist die Wahrscheinlichkeit groß, dass sie sofort wieder eingelagert werden muss. Die zuvor eingelagerte Instanz dagegen wird höchstwahrscheinlich gleich wieder zurückgeschrieben, wenn sie in der nächsten Zeit nicht wieder angefasst wird. Die häufigen Zugriffe auf die Platte kosten Zeit. Ein Ziel ist es deshalb, unnötige Plattenzugriffe zu vermeiden. Dies kann erreicht werden, indem eine ausgelagerte Instanz solange von der Migration zurückgestellt wird, bis sie z. B. wegen Weiterschalten sowieso eingelagert werden muss.

Kapitel 5

Nebenläufigkeit

Prozess-Management-Systeme zählen zu den Mehrbenutzer-Systemen. Es können u. U. mehrere hundert Benutzer gleichzeitig angemeldet sein und das PMS in Anspruch nehmen. Die Aktionen, die sie ausführen, können sich ohne weitere Kontrollen durch das System beeinflussen oder sogar stören. In einem adaptiven PMS können z. B. mehrere Benutzer versuchen, dieselbe Prozessvorlage zur gleichen Zeit anzupassen oder diesselbe Instanz dynamisch abzuändern. Häufiger wird der Fall auftreten, dass Benutzer Instanzen weiterschalten, die von anderen Benutzern gerade modifiziert werden oder die auf Vorlagen basieren, die angepasst werden müssen. Welche Probleme dabei entstehen und wie sie verhindert werden können, soll in diesem Kapitel untersucht werden. Konkurrierende Zugriffe einfach zu verbieten, kann u. U. zu restriktiv sein.

Aber nicht nur Benutzeraktionen untereinander, sondern auch Benutzer- und Systemaktionen können sich beeinflussen. Eine für diese Arbeit relevante Systemfunktion ist die Migration, die mit Ausnahme des Benutzereingriffs bei Konflikten automatisch abläuft. Das Weiterschalten oder die Modifikationen von in Migration befindlichen Instanzen oder das erneute Abändern einer Vorlage, auf die noch immer Instanzen aus einer vorherigen Schemaevolution zu migrieren sind, sind in diesem Zusammenhang zu untersuchen.

Zusammengefasst müssen folgende Aktionen in ihren Wechselwirkungen untersucht werden:

1. Vorlagenänderung ↔ Vorlagenänderung
2. Vorlagenänderung ↔ Weiterschalten
3. Vorlagenänderung ↔ Dynamische Änderungen
4. Dynamische Änderungen ↔ Weiterschalten
5. Dynamische Änderungen ↔ Dynamische Änderungen
6. Migration ↔ Weiterschalten

7. Migration ↔ Vorlagenänderung
8. Migration ↔ Dynamische Änderungen

5.1 Vorlagenänderung ↔ Vorlagenänderung

Eine Situation, bei der es zu Konflikten aufgrund von konkurrierenden Zugriffen kommen kann, liegt vor, wenn zwei oder mehrere Benutzer versuchen, gleichzeitig dieselbe Vorlage zu ändern.

Die Konflikte, die auftreten können, entsprechen den in Kapitel 2.3.2 beschriebenen auftretbaren Konflikten zwischen Instanz- und Vorlagenänderungen: Die Benutzer können sich widersprechende Modifikationen an der Vorlage vornehmen, sie können versuchen, an dieselbe Stelle unterschiedliche Aktivitäten einzufügen oder sie können Änderungen durchführen, die für sich genommen zwar zu einem wiederum korrekten Schema führen, zusammen aber ein inkorrektes Schema ergeben, wie es z. B. in Zusammenhang mit dem Einfügen von Sync-Kanten vorkommen kann.

Ziel ist es deshalb, dass Konflikte aufgrund von überlappenden Änderungen vermieden oder aufgelöst werden.

Dieses Ziel kann durch unterschiedliche Verfahren erreicht werden. Mögliche Verfahren sind:

- Exklusivzugriff
- Bestätigung einer Änderung durch den Server
- Sperren
- Optimistisches Verfahren

Diese werden im Folgenden erläutert.

Exklusivzugriff

Die intuitivste und einfachste Lösungsstrategie zum Erreichen des Ziels ist es, das gleichzeitige Abändern ein und derselben Vorlage durch mehrere Benutzer völlig zu unterbinden, d. h. dem Benutzer, der zuerst die Änderung der Vorlage beim Server beantragt, Exklusivzugriff in Bezug auf Änderungen zu gewähren. Der Ablauf der Vorlagenänderung entspricht dem in Kapitel 2.1.2 über das Konzept der Schemaevolution beschriebenen Ablauf.

Der Vorteil eines Exklusivzugriffes liegt darin, dass keine zusätzlichen Datenstrukturen oder aufwändige Verfahren notwendig sind, um Konflikte zu verhindern oder aufzulösen, da aufgrund des Exklusivzugriffes keine diesbezüglichen Konflikte auftreten können. Weiter besteht für den Benutzer die Möglichkeit, die Änderungen offline vorzunehmen, wenn er bei Beginn der

Änderungstransaktion die Kopie der Vorlage auf seinen Rechner überspielt bekommt. Er kann diese dann ohne ständigen Kontakt zum Server entsprechend seiner Wünsche anpassen. Bei Commit wird die angepasste Kopie dann zurück an den Server geschickt und als neue Version der Prozessvorlage ins PMS übernommen. Oder es werden nur die angewendeten Operationen dem Server mitgeteilt, die der Server dann auf seiner Kopie des Vorlagen-Objekts anwendet. Dieses Vorlagen-Objekt repräsentiert schließlich die neue Version der Prozessvorlage. Bis nach Abschluss der Commit-Prozedur bleibt es den anderen Benutzer versagt, dieselbe Vorlage zu ändern.

Dieses Verfahren kann aber zu restriktiv sein, besonders in der Entwicklungszeit, wenn mehrere Personen an der Gestaltung des Geschäftsprozesses beteiligt sind und häufig eventuell lang andauernde Erweiterungen des bestehenden Schemas vornehmen müssen. Die Arbeit zurückzustellen bis der Kollege seine Modellierungen abgeschlossen hat, ist im Allgemeinen nicht tragfähig. Werden keine weiteren Vorsichtsmaßnahmen getroffen, wie z. B. ein Zeitlimit für den Exklusivzugriff, dann kann es zu unnötigen Blockaden aufgrund nicht freigegebener Vorlagen kommen, z. B. wenn der Benutzer zwar mit den Änderungen fertig ist, aber mit dem Einspielen noch wartet, oder die Änderungen verwirft, ohne das System davon zu benachrichtigen, oder wenn aufgrund von Kommunikationsproblemen das Zurückspielen auf den Server nicht erfolgen kann.

Die im folgenden genannten Verfahren zur Verhinderung oder Vermeidung von Konflikten ermöglichen das gleichzeitige – aber eventuell eingeschränkte – Abändern eines Schemas durch mehrere Benutzer, sorgen aber dafür, dass das oben genannte Ziel trotzdem erfüllt wird.

Bestätigung einer Änderung durch den Server

Eine Möglichkeit, paralleles Abändern einer Vorlage zu gewähren, ohne aber Konflikte aufkommen zu lassen, besteht darin, den Server über die Anwendung einer jeden Änderungsoperation entscheiden zu lassen. Dabei schickt jeder Benutzer seine Änderungswünsche an den Server. Dieser nimmt sie in der Reihenfolge entgegen, wie sie bei ihm eintreffen, und untersucht für jeden einzelnen Wunsch, ob er erfüllt werden kann, d. h. ob er nicht mit den Änderungswünschen anderer kollidiert. Kann dem Wunsch entsprochen werden, so führt er die entsprechende Operation auf der Kopie der Vorlage aus. Zusätzlich schickt er an alle für die Anpassung der betroffenen Vorlage registrierten Clients eine Bestätigung über die Anwendung dieser Operation. Auf diese Weise erfährt der Client, der die entsprechende Änderungsoperation durchführen wollte, dass sie angenommen wurde. Alle anderen Clients werden dadurch ständig über die bereits erfolgten Änderungen der anderen auf dem Laufenden gehalten und können entsprechend ihre Darstellung aktualisieren. Kann die Operation aufgrund von Konflikten nicht angewendet werden, so muss nur der Absender davon unterrichtet werden. Er kann dann den Benutzer mit einem entsprechenden Hinweis informieren, welche Operation abgelehnt wurde und warum.

Client 1 aus Abbildung 5.1 möchte in den dargestellten Prozessgraph die Sync-Kante $A12 \rightarrow A21$ einfügen. Dazu stellt er eine entsprechende Anfrage an den Server. Dieser überprüft, ob das Einfügen der Sync-Kante in das Schema Konflikte verursachen würde. Da dies nicht der Fall ist,

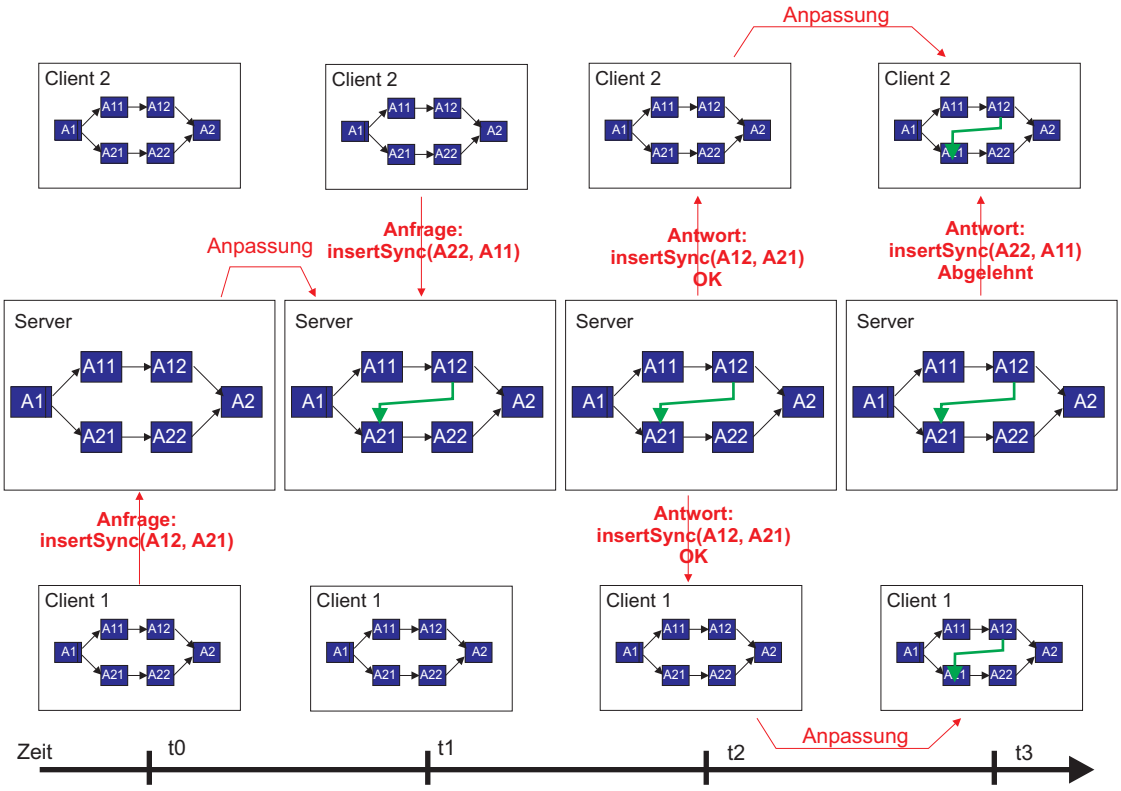

Abbildung 5.1: Prinzip der Bestätigung einer Vorlagenänderung durch den Server

gibt er an alle Clients, welche am Ändern des Schemas beteiligt sind, eine Bestätigungsmeldung heraus, nachdem er selbst die Änderung auf seiner Kopie der Vorlage durchgeführt hat. Die Clients passen daraufhin ihre Darstellung entsprechend an. Bevor Client 2 jedoch von der neuen Sync-Kante erfährt, beantragt er die Einfügung der Sync-Kante $A22 \rightarrow A11$. Diese Operation wird jedoch vom Server abgelehnt, da diese Sync-Kante zusammen mit der zuvor eingefügten zu einem Zyklus im Prozessgraphen führen würde.

Ein Vorteil dieses Verfahrens ist neben der Möglichkeit, dass mehrere Benutzer das Schema parallel abändern können, dass alle Benutzer auch sofort – sofern keine Bestätigung verloren geht – über die gemachten Änderungen der anderen informiert werden. Dieses Verfahren ist auch sehr gut für eine Umgebung geeignet, in denen die Notwendigkeit für „Thin Clients“ besteht: Der Server übernimmt die vollständige Prüfung auf Korrektheit des Schemas und auf das Fehlen von Konflikten aufgrund von überlappenden Änderungen. Der Client muss somit diese Funktionalität nicht bereitstellen, sondern konzentriert sich ausschließlich auf die Visualisierung des Prozessgraphen.

Dass der Server sämtliche Überprüfungen durchführen muss, kann aber zu Zeiten hoher Last, oder wenn viele Vorlagen gleichzeitig abgeändert werden, zu starken Leistungseinbußen des Sys-

tems führen. Ein weiterer Nachteil ist, dass das Versenden von Änderungsanforderungen und Änderungsbestätigungen einen erheblichen Kommunikationsaufwand verursacht. In stark beanspruchten Netzen oder über mehre Teilnetze hinweg kann es dann zu erheblichen Verzögerungen kommen. Auch muss der PMS-Server ständig erreichbar sein, um eine Vorlage abändern zu können. Es ist bei diesem Verfahren somit nicht möglich, offline zu arbeiten. Weitere Probleme treten auf, wenn Bestätigungsmeldungen verloren gehen. Die Clients können dann nicht die entsprechenden Operationen auf ihrem Schema nachziehen. Es kann dann im folgenden zu Fehlern kommen, wenn Bestätigungen über Änderungsoperationen eintreffen, die auf der fehlenden Modifikation beruhen. Auch die Ankunft der Bestätigungen in falscher Reihenfolge aufgrund von unterschiedlich zurückgelegten Routen der Nachrichten im Netzwerk führt zu diesem Fehler. Der größte Nachteil aber ist, dass das Rückgängigmachen einer schon bestätigten Änderung sehr aufwändig sein kann, nämlich dann, wenn andere Benutzer bereits anderweitige, darauf basierende Modifikation ausgeführt haben. Somit ist es schwer, Änderungstransaktionen zu realisieren, die abgebrochen werden können.

Eine Frage bleibt noch zu klären, die in Zusammenhang mit diesem Verfahren auftritt: Wann ist der Zeitpunkt gekommen, an dem eine neue Version der Vorlage zu existieren beginnt? Der einzig sinnvolle Zeitpunkt ist dann, wenn alle beteiligten Benutzer ihre Modifikationen abgeschlossen haben. Dabei besteht allerdings die Gefahr, dass sich die Versionierung lange Zeit hinauszögert, erstens wenn immer wieder neue Benutzer die Vorlage anzupassen beabsichtigen und damit den Abschluss verhindern und zweitens wenn die Modifikationen eines oder mehrerer Benutzer lange Zeit in Anspruch nehmen. Dem ersten Fall kann begegnet werden, indem die Zahl der Personen beschränkt wird, welche die Vorlage abändern können, ohne dass eine neue Version der Vorlage zustande kommt. Überschreitet die Zahl der änderungswilligen Personen die festgelegte Zahl, so müssen diese warten, bis alle bereits zugelassenen Benutzer ihre Modifikationen abgeschlossen haben und damit eine neue Version des Schemas angelegt werden konnte.

Sperrverfahren

Beim Sperrverfahren werden Prozessabschnitte, die abgeändert werden, gegen den Zugriff durch andere gesperrt. Es kann dadurch nicht nur Konflikten vorgebeugt werden, es wird damit auch verhindert, dass Modifikationen eines Benutzers auf Modifikationen eines anderen aufbauen. Das Verfahren zielt darauf ab, die Schemaänderungen durch die verschiedenen Benutzer disjunkt zu halten. Dadurch ist es besonders einfach, einzelne oder alle Änderungen eines Benutzers wieder rückgängig zu machen. Dieses Verfahren unterstützt damit Änderungstransaktionen. Die entsprechend gesetzten Sperren müssen beim Rückgängigmachen aufgehoben werden.

Die Benutzer ändern das Schema auf einer lokal liegenden Kopie der Vorlage ab. Bei einem Commit werden die entsprechenden Änderungen wie in Kapitel 2.1.2 dargelegt ins PMS übernommen und als neue Version der Vorlage dort hinterlegt. Welcher Benutzer zuerst commited spielt keine Rolle, da die Änderungen der verschiedenen Benutzer durch dieses Verfahren disjunkt gehalten werden und disjunkte Änderungen nach [Ri04] kommutativ sind. Wird die neue Version der Vorlage erneut abgeändert, noch bevor alle parallel dazu gemachten Änderungen

eingebracht wurden, so müssen die bereits existierenden Sperren auch dabei noch berücksichtigt werden. Dies ist notwendig, damit auch die restlichen Änderungen noch problemlos eingespielt werden können, ohne dass es zu Überlappungen mit den neuen Modifikationen kommt.

Vor jeder Anwendung einer Operation muss ähnlich dem vorigen Verfahren überprüft werden, ob die Änderungen erfolgen dürfen, indem Exklusivsperren auf den Bereich angefordert werden, der von der anzuwendenden Operation betroffen ist. Können die Sperren nicht gewährt werden, da die Bereiche oder Teile davon bereits Exklusivsperren durch andere Benutzer aufweisen, dann darf die Operation nicht angewendet werden. Für jeden Typ von Änderungsoperation kann ein entsprechender Sperrbereich definiert werden. Das Beispiel aus Abbildung 5.2 demonstriert das Prinzip des Sperrverfahrens:

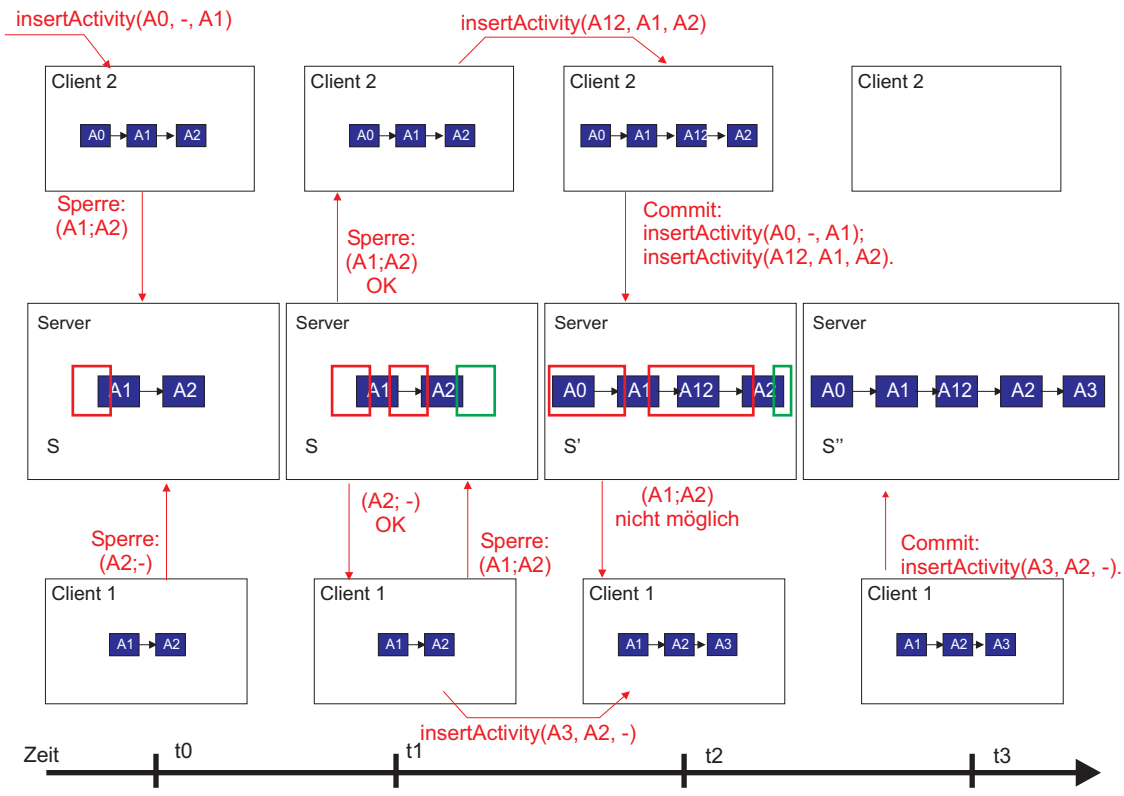

Abbildung 5.2: Prinzip des Sperrverfahrens

In dem Beispiel ändern Client 1 und Client 2 beide parallel das Schema S ab. Client 2 hat zuerst die Aktivität A0 eingefügt. Dazu hatte er die Sperre (-;A1) beantragt, die auch gewährt werden konnte. Um dann noch die Aktivität A12 zwischen A1 und A2 einfügen zu können, beantragt Client 2 zusätzlich zum Zeitpunkt t0 die Sperre auf den Bereich (A1, A2), damit kein anderer Benutzer an dieser Stelle eine andere Aktivität einfügen oder A1 bzw. A2 verschieben kann. Zeitgleich beantragt Client 1 die Sperre auf (A2; -), um hinter A2 die Aktivität A3 einfügen

zu können. Beide Sperren können gesetzt werden. Somit können beide Clients ihre gewünschten Änderungen durchführen. Client 1 will danach zusätzlich noch zwischen A1 und A2 eine Aktivität einfügen. Die dazu notwendige Sperre (A1; A2) bekommt er aber nicht, da diese bereits von Client 2 gehalten wird. Somit kann er diese Änderungsoperation auf dem Schema nicht vornehmen. Die Sperren von Client 2 werden auch nach dessem Commit seiner Änderungstransaktion aufrechterhalten, damit Client 1, der seine Modifikationen noch nicht abgeschlossen hat, nicht im Nachhinein noch Änderungen an den von Client 2 bereits modifizierten Prozessabschnitten vornehmen kann, das zu Konflikten beim Einspielen der Änderungen von Client 1 auf den Server führen könnte. Da die Sperren für disjunkte und konfliktfreie Modifikationen zwischen mehreren Benutzern sorgt, können die Änderungsoperationen von Client 1 bei Abschluss seiner Änderungstransaktion ohne weitere Prüfungen auf den Server eingespielt werden. Nach dem Commit von Client 1 werden sämtliche Sperren wieder freigegeben, da keine weiteren Clients mehr existieren, die dieses Schema bearbeiten.

Die Sperren müssen in einer Sperrtabelle verwaltet werden, die auf einem zentralen Server liegt. Jeder änderungswillige Client muss darauf Zugriff haben. Die Granularität der Sperrbereiche bestimmt die Größe der Tabelle und die Häufigkeit des Zugriffs darauf. Da sich oft die Änderungen in einem Bereich kleiner Ausdehnung konzentrieren, ist es denkbar, präventiv einen größeren Ausschnitt zu sperren, als es die bereits ausgeführten Änderungsoperationen eigentlich erfordern. Die Sperrtabelle muss dann nicht aktualisiert werden, wenn eine weitere Modifikation des Schemas in einen der bereits gesperrten Bereiche fällt.

Der Aufwand für die Verwaltung einer Sperrtabelle zu diesem Zweck hält sich allerdings in Grenzen, da im Allgemeinen relativ wenig Benutzer gleichzeitig ein und dieselbe Vorlage abändern und relativ wenig Änderungsoperationen ausgeführt werden, verglichen mit der Häufigkeit, mit der Aktivitäten weitergeschalten werden.

Von Nachteil bei diesem Verfahren ist, dass der Server mit der Sperrtabelle ständig erreichbar sein muss. Damit ist dieses Verfahren auch nicht für die Offline-Arbeit geeignet.

Optimistisches Verfahren

Beim optimistischen Verfahren modifiziert jeder Benutzer seine eigene Kopie der Vorlage wie beabsichtigt. Er braucht dabei erst einmal keine Rücksicht auf die Änderungen der anderen Benutzer zu nehmen. Beim Commit wird entweder die ganze Kopie zurück an den Server gesendet oder es werden ihm nur die angewendeten Änderungsoperationen mitgeteilt. Ist die momentan aktuelle Version der Vorlage genau die, auf dem der Benutzer seine Änderungen aufgebaut hat, so wird sein Schema, sofern korrekt, als neue Version der Vorlage ins PMS übernommen. Hat ein anderer Benutzer jedoch bereits seine Modifikationen eingespielt, d. h. die aktuell relevante Version stimmt nicht mit der überein, auf dem die jetzt einzuspielenden Modifikationen beruhen, dann muss der Server den Überlappungsgrad (vgl. Kapitel 2.3.2) der beiden Vorlagenänderungen bestimmen und darauf basierend analog zur Migration von gegenüber der Vorlage verzerrten Instanzen das resultierende Schema erzeugen, welches daraufhin als neue Vorlagenversion ins

PMS übernommen wird. Zuvor muss der Server aber überprüfen, ob das Einspielen der neu hinzugekommenen Änderungen auf das momentan relevante Schema nicht zu einem inkorrekten Schema führt. Im Falle von Konflikten unterrichtet er den entsprechenden Benutzer und gibt ihm die Möglichkeit, sie zu beheben. Für den Test auf Konflikte und bei der Bestimmung des Überlappungsgrades der Änderungen können dieselben Algorithmen angewendet werden wie bei der Migration von verzerrten Instanzen.

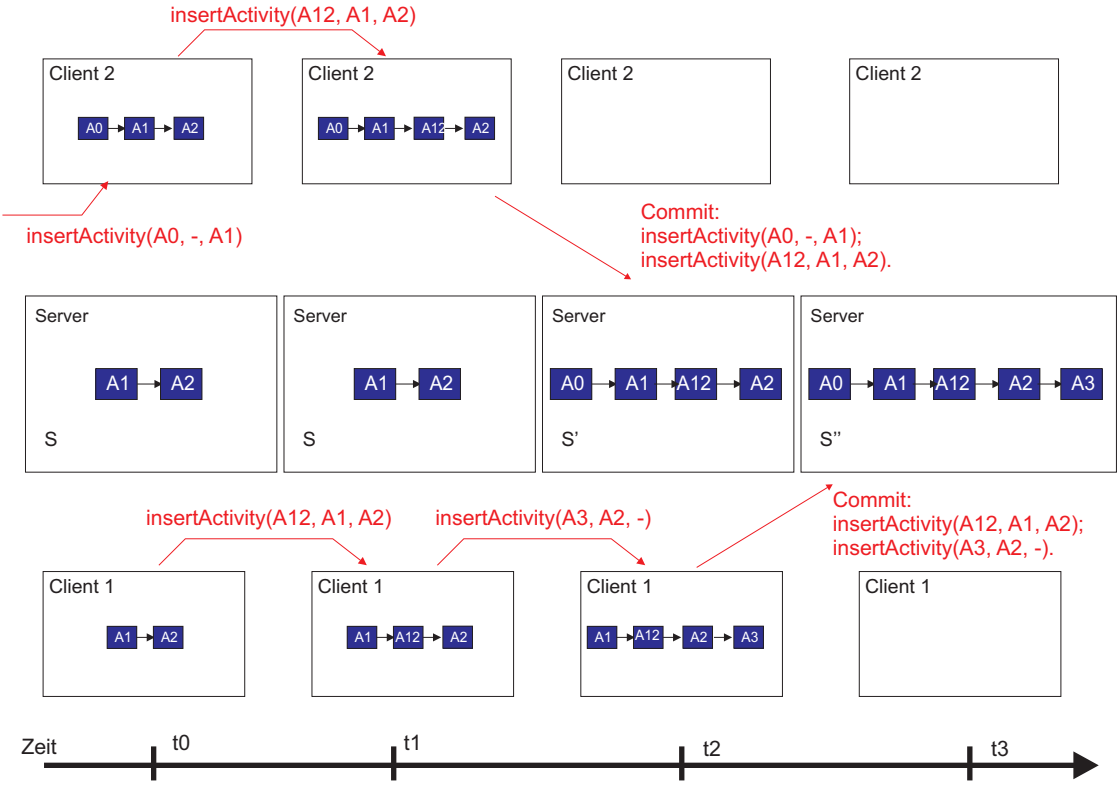

Abbildung 5.3: Prinzip des optimistischen Verfahrens

In dem Beispiel aus Abbildung 5.3 ändern Client 1 und Client 2 parallel und unabhängig voneinander das Schema S ab. Client 2 commited als erster. Da die Version des auf dem Server vorliegenden Schemas noch mit der Version des Schemas übereinstimmt, auf dem Client 2 seine Modifikationen begonnen hat, können seine Änderungen einfach auf den Server eingespielt werden. Zum Commit-Zeitpunkt von Client 1 sieht der Fall anders aus: Die aktuelle Version S' des Schemas auf dem Server stimmt nicht mehr mit der Version S überein, die Client 1 begonnen hat abzuändern. Aus diesem Grund muss nun der Überlappungsgrad der von Client 1 durchgeführten Änderungen mit den Änderungen bestimmt werden, die das auf dem Server vorliegende Schema von der Version S in die Version S' überführt haben, d. h. in diesem Fall mit den Änderungen von Client 2. Da beide Clients die Aktivität A12 eingefügt haben, ansonsten aber keine gemeinsame Aktivitäten dem ursprünglichen Schema hinzugefügt haben, liegt der

einfache Fall der partiellen Äquivalenz vor. Um die Änderungen von Client 1 in dem serverseitig vorliegenden Schema zu berücksichtigen, ist es deshalb nur nötig, die neue Aktivität A3 dort einzubringen.

Der Vorteil dieses Verfahrens ist, dass jeder Benutzer erst einmal nicht in seiner Modellierungsfreiheit eingeschränkt ist. Er kann sämtliche Modifikationen vornehmen, die er geplant hat und wie er sie geplant hat. Erst im Falle von Konflikten muss er sich darüber Gedanken machen, wie er diese umgehen kann. Dabei kann er sinnvoller als bei den anderen Verfahren vom Server unterstützt werden, weil der Server bereits die Ziele des Modellierers in Form des fertigen Schemas kennt. Dadurch, dass kein ständiger Kontakt zum Server erforderlich ist, ist das Offline-Modellieren möglich. Da die Benutzer unabhängig voneinander auf getrennten Vorlagenkopien ihre Modifikationen vornehmen, können einzelne Änderungen auch wieder rückgängig gemacht werden, ohne dass es zu Konflikten aufgrund von darauf basierenden Änderungsoperationen anderer Benutzer kommt. Das Abbrechen einer ganzen Änderungstransaktion kann auch leicht realisiert werden, indem einfach die abgeänderte Vorlagenkopie verworfen wird. Ein weiterer Vorteil ist, dass keine zentrale Datenstruktur, wie z. B. eine Sperrtabelle, notwendig ist, die zum Nadelöhr werden kann.

Der Nachteil ist, dass das Einspielen der Änderungen verglichen mit den anderen, bis jetzt vorgestellten Verfahren ein relativ aufwändiger Vorgang ist, da die Algorithmen zur Bestimmung des Überlappungsgrades auf Mengenvergleichen (vgl. [Ri04]) beruhen.

Welches der vorgestellten Verfahren zu bevorzugen ist, kann nicht pauschal gesagt werden, sondern hängt von den gegebenen Umständen ab: Sind konkurrierende Änderungen der Vorlage so gut wie ausgeschlossen oder nicht notwendig, dann ist das Verfahren des Exklusivzugriffs zu bevorzugen, da es weder großen Verwaltungsaufwand für das System verursacht, noch komplexe Algorithmen notwendig sind, um die konkurrierende Änderungen zusammenzubringen, wie es beim optimistischen Verfahren der Fall ist. Da in Unternehmen in der Regel die Zuständigkeiten für logisch in sich abgeschlossene Prozessabschnitte klar verteilt sind, ist die Wahrscheinlichkeit sehr gering, dass zwei Personen denselben Abschnitt abändern. Dann bietet sich das Sperrverfahren oder das optimistische Verfahren an.

5.2 Vorlagenänderung ↔ Weiterschalten

In Zusammenhang mit einer Vorlagenänderung muss auch untersucht werden, ob und inwieweit es zu Konflikten kommen kann, wenn Instanzen, die auf dieser Vorlage basieren, zeitgleich weiterschalten.

Da die Vorlagenänderungen auf einer oder mehreren Kopien der momentan gültigen Vorlagenversion durchgeführt werden, kommt es primär zu keinen Problemen, wenn eine auf dieser Vorlage basierende Instanz zeitgleich weitergeschaltet wird: Das diese Instanz repräsentierende Instanz-Objekt bzw. – im Falle einer verzerrten Instanz – das zugehörige unterste Delta-Schicht-Objekt

referenziert vorerst weiterhin das originale Vorlagen-Objekt. Die Bestimmung der als nächstes zu aktivierenden Aktivitäten erfolgt deshalb weiterhin problemlos nach der bisherigen Version.

Allerdings kann das Weiterschalten die Instanz in einen Zustand bringen, der dann nicht mehr mit den Änderungen des Schemas verträglich ist, wie Abbildung 5.4 zeigt.

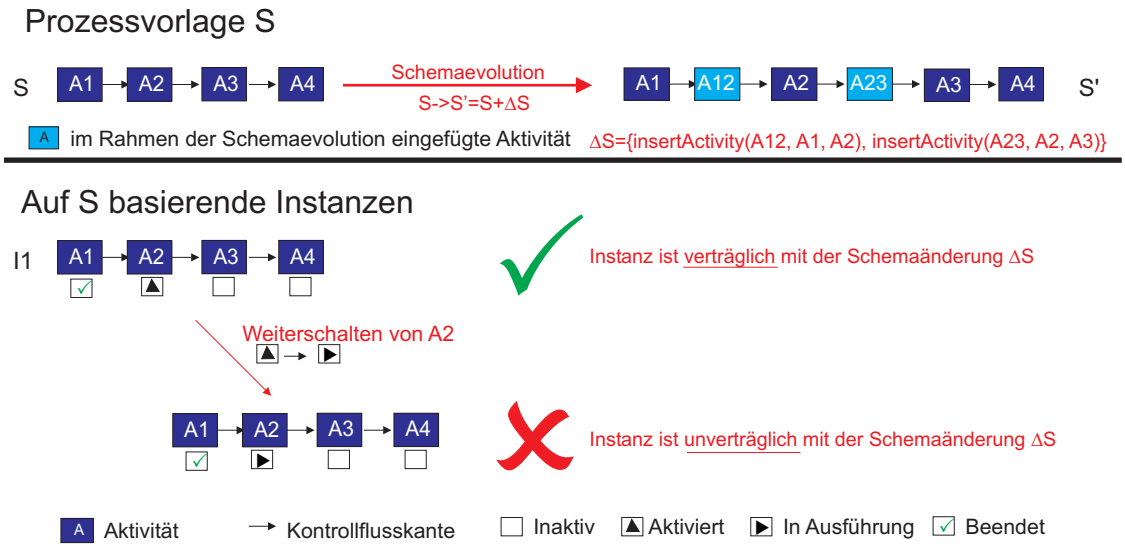

Abbildung 5.4: Unverträglichkeit einer Instanz aufgrund einer Aktivitäten-Weiterschaltung

Hierbei kann das Sperren von Instanzen gegen das Weiterschalten schon während der Änderungsphase verhindern, dass Instanzen in Zustände gelangen, in denen sie nicht mehr verträglich mit den Änderungen sind.

Zwei Sperransätze sind denkbar. Ein Ansatz ist, die Instanz komplett gegen das Weiterschalten zu sperren. Ein anderer Ansatz ist, nur die Aktivitäten gegen das Weiterschalten zu sperren, die später von der Verträglichkeitsprüfung auf ihren Status hin untersucht werden. Wie in Kapitel 2.3.3 über verträgliche und unverträgliche Instanzen angedeutet, hängt es von den durchgeführten Modifikationen ab, welche Aktivitäten untersucht werden müssen. Die zu sperrenden Aktivitäten lassen sich somit anhand der angewendeten Änderungsoperationen bestimmen.

Der zweite Ansatz hat gegenüber dem ersten Ansatz den Vorteil, dass nur Aktivitäten gegen das Ausführen gesperrt werden, die auch wirklich von der Verträglichkeitsprüfung betroffen sind. Das Weiterschalten der anderen Aktivitäten wird dabei nicht behindert.

Da aber erst nach der Anwendung einer Änderungsoperation bekannt ist, welche Aktivitäten davon betroffen sind, kann dieser Ansatz nur verhindern, dass die Instanz in einen unverträglichen Zustand gebracht wird, indem die bereits bestimmten Aktivitäten ausgeführt werden. Die Instanz kann aber immer noch durch das Weiterschalten der anderen Aktivitäten einen unverträglichen Zustand einnehmen, nämlich dann, wenn im Laufe der Transaktion eine Operation

angewendet wird, bei deren Prüfung auf Anwendbarkeit im Verträglichkeitstest des Migrationsprozesses die Überprüfung des Status einer dieser Aktivitäten notwendig ist.

Die Abbildungen 5.5 und 5.6 verdeutlichen dies:

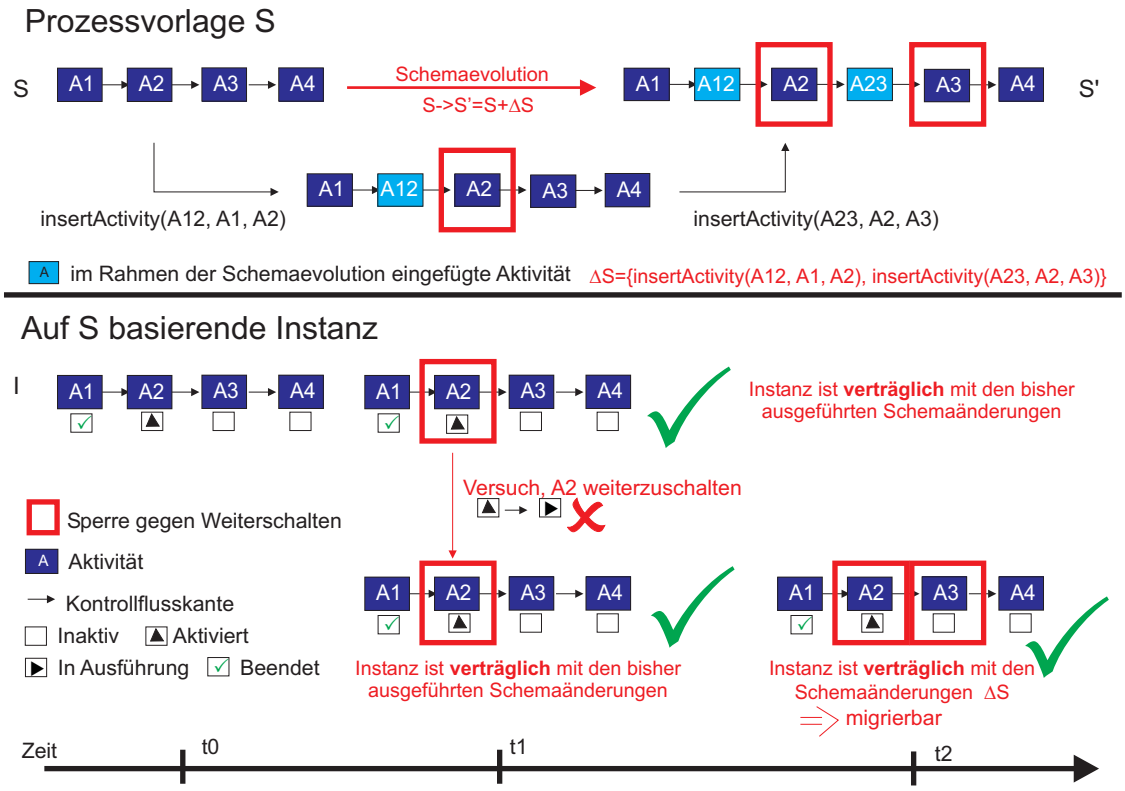

Abbildung 5.5: Vermeidung von Unverträglichkeit durch Sperren gegen Weiterschalten

Bei Abbildung 5.5 vereitelt die nach der Schemaänderung $insertActivity(A12,\ A1,\ A2)$ auf A2 gesetzte Sperre den zum Zeitpunkt t1 stattfindenden Versuch, A2 zu starten, und verhindert damit, dass die Instanz I unverträglich mit den Schemaänderungen ΔS wird.

Wird aber, wie in Abbildung 5.6 gehandhabt, anstatt A12 zuerst die Aktivität A23 eingefügt, dann existiert zum Zeitpunkt t1 die Sperre auf A2 noch nicht und der Versuch gelingt, A2 zu diesem Zeitpunkt zu starten. Dies hat zur Folge, dass I mit dem Einfügen von A12 in das Vorlagenschema zum Zeitpunkt t2 trotz des Einsatzes von Sperren unverträglich mit den Schemaänderungen ΔS wird und im Folgenden nicht migriert werden kann. Die Sperre A12 existiert bei t1 noch nicht, da das System zu diesem Zeitpunkt noch nicht wissen kann, dass zu einer späteren Zeit noch die Aktivität A12 in das Schema eingefügt werden soll.

Die Instanz komplett gegen das Weiterschalten zu sperren, blockiert zwar alle Aktivitäten, aber es kann dabei nur der bereits zu Beginn der Schemaänderung eingenommene Zustand der Instanz

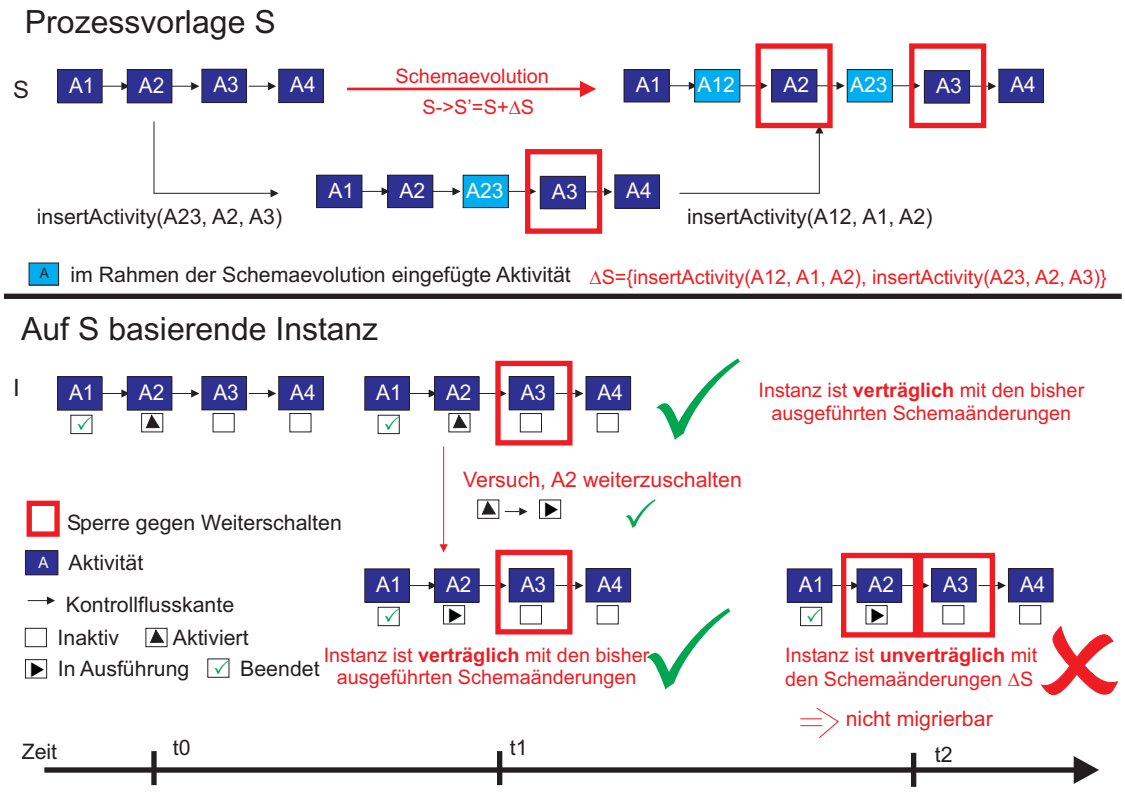

Abbildung 5.6: Unverträglichkeit trotz Sperren gegen Weiterschalten

verantwortlich für die Unverträglichkeit mit den Schemaänderungen sein. Nachträglich in einen solchen Zustand zu laufen ist dabei unmöglich.

Der Einsatz von Sperren schon während der Vorlagenänderung weist allerdings erhebliche Nachteile auf:

Das Sperren kann die Instanz unnötigerweise oder unnötig lange blockieren, erstens wenn die Vorlagenänderung abgebrochen wird, zweitens wenn die Instanz aufgrund ihres bereits erlangten Zustandes schon unverträglich ist oder es im Laufe der Änderungstransaktion wird, und drittens wenn die Instanz noch vor dem Abschluss der Schemaevolution beendet werden könnte.

Werden nur einzelne Aktivitäten gegen das Weiterschalten gesperrt, so ist eine Sperrtabelle notwendig, die entsprechende Informationen bereithält. Da sehr viele Instanzen eines Typs existieren können und jedes Weiterschalten ein Zugriff auf die Tabelle erforderlich macht, kann die Datenstruktur zu einem Engpass werden. Verzögerungen bei den Prozessausführungen sind die Folge.

Zusätzlich von Nachteil ist, dass dabei das Offline-Ändern der Vorlage nicht mehr möglich ist, auch wenn das eingesetzte Verfahren zur Verhinderung von konkurrierenden Vorlagenänderungen

(vgl. Abschnitt 5.1) dies zulassen würde, da nach jeder Anwendung einer Änderungsoperation die entsprechenden Sperren auf dem Server gesetzt werden müssen. Dazu ist der ständige Zugriff aller an der Schemaänderung beteiligten Clients auf den Server notwendig.

Einfach die entsprechenden Arbeitsaufträge aus den Arbeitslisten der infrage kommenden Bearbeiter herauszunehmen reicht im Allgemeinen zur Verhinderung einer Weiterschaltung nicht aus, besonders nicht, wenn die Clients selbst für die Aktualisierung ihrer Arbeitslisten verantwortlich sind: Einige der Clients können das Löschen der Einträge aufgrund zu langer Aktualisierungsintervalle zu spät oder gar nicht mitbekommen. Deshalb muss der Server trotzdem noch bei einer Auftragsannahme gesondert überprüfen, ob die jeweilige Aktivität gestartet werden darf.

Zusammenfassend kann gesagt werden, dass Sperren auf der einen Seite verhindern können, dass eine mit den Schemaänderungen verträgliche Instanz noch unverträglich wird, auf der anderen Seite aber auch Instanzen lange blockieren können. Die Vor- und Nachteile eines Einsatzes sollten deshalb gut gegeneinander abgewägt werden. Die Gründe für die Schemaänderungen können u. U. die Entscheidung beeinflussen. Sind Optimierungen der Anlass der Änderungen, so ist im Allgemeinen das Sperren nicht notwendig. Beseitigen die Schemaänderungen allerdings Fehler im Prozessablauf, so ist das Sperren der Instanzen im Allgemeinen unumgänglich. Auch die Ausführungsbrisanz muss in die Überlegungen mit einbezogen werden. Bei Leasing-Verträgen zum Beispiel, bei denen es bei dem Ausführungszeitpunkt der einzelnen Schritte nicht auf das Heute oder auf das Morgen ankommt, kann dem Sperren eher zugestimmt werden, als bei Prozessen, bei denen strenge Zeitanforderungen vorliegen.

5.3 Vorlagenänderung ↔ Dynamische Änderungen

Weitere, auf Wechselwirkung hin zu untersuchende konkurrierende Aktionen sind die Vorlagenänderung und das dynamische Ändern einer Instanz, die auf dem entsprechenden Schema basieren.

Von der Architektur her gibt es dabei keine Probleme: Das Delta-Schicht-Objekt, in dem die Änderungen der Instanz gegenüber der Vorlage gespeichert werden, setzt direkt oder transitiv über andere Delta-Schichten auf dem originalen Vorlagen-Objekt auf, das die bisher gültige Version des Schemas repräsentiert. Die Schemaänderungen werden dagegen auf einer Kopie dieses Vorlagen-Objekts ausgeführt, wie in Kapitel 3.3 beschrieben. Damit kommt es auf dieser Ebene zu keinen Wechselwirkungen zwischen den Änderungsarten.

Bei der Migration der dynamisch geänderten Instanz auf das neue Schema kann es aber zu Konflikten aufgrund partiell äquivalenten Instanz- und Vorlagenänderungen kommen, wie in Kapitel 2.3.2 über die Migration verzerrter Instanzen geschildert wurde.

Zum Beispiel kann an dieselbe Stelle auf Typebene eine andere Aktivität als auf Instanzebene eingefügt werden. Im Rahmen der Migration der Instanz ergeben sich dann mehrere denkbare Varianten, wie sich die Instanz auf die neue Version der Vorlage anpassen ließe, wie Abbil-

dung 5.7 zeigt. Welche Variante die richtige ist, muss mithilfe des Benutzers oder speziell dafür hinterlegten Regeln beantwortet werden.

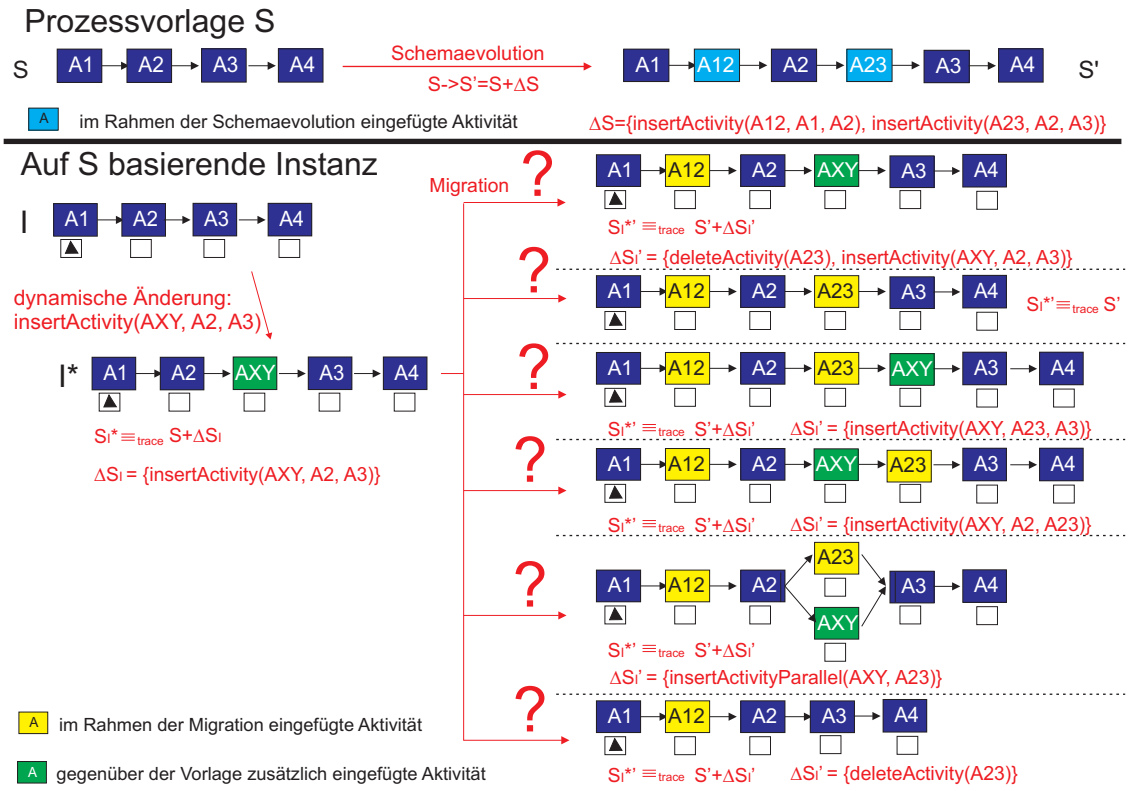

Abbildung 5.7: Konflikte aufgrund überlappender Instanz- und Schemaänderungen

Ziel dieses Abschnitts ist zu klären, ob es sinnvoll ist, dynamische Änderungen einer Instanz zu untersagen, während das zugrunde liegende Schema abgeändert wird, und ob es andere Möglichkeiten gibt.

Instanzen während der Schemaevolution gegen das dynamische Ändern zu sperren verhindert, dass es bei der Migration dieser Instanzen zu Problemen aufgrund von überlappenden oder konkurrierenden Änderungen kommt, vorausgesetzt die Instanzen sind nicht schon bereits gegenüber der momentan gültigen Schemaversion verzerrt. Benutzereingriffe während der Migration können dadurch reduziert werden.

Die Instanz ganz gegen das dynamische Abändern zu sperren, muss aber nicht unbedingt sinnvoll sein: Wie in Abschnitt „Adaptivität“ der Einführung (Abschnitt 1.2) erklärt, werden dynamische Änderungen u. a. angewendet, um Ausnahmesituationen zu begegnen. Wird die Instanz in so einer Situation gegen das Ändern gesperrt, dann kann eventuell nicht rechtzeitig auf die Ausnahme reagiert werden. Unter Umständen kann daraus mehr Schaden als Nutzen entstehen.

Eine praktikable Lösung ist es, einfach die Benutzer von der anstehenden Schemaänderung in Kenntnis zu setzen und ihnen die Entscheidung zu überlassen, ob sie die Instanzen trotzdem modifizieren wollen.

Wird zur Kontrolle von konkurrierenden Vorlagenänderungen das in Abschnitt 5.1 beschriebene Sperrverfahren eingesetzt, das die Schemaänderungen der verschiedenen Benutzer disjunkt hält, dann kann durch die Berücksichtigung der dabei angefallenen Sperrinformationen die Wahrscheinlichkeit von Konflikten reduziert werden, ohne die Instanz ganz gegen dynamische Änderungen während der Schemaänderung zu sperren.

Die Sperren zu berücksichtigen kann die Überlappung der Instanzänderungen mit den Schemaänderungen reduzieren, da auf Instanzebene keine Modifikationen durchgeführt werden können, die Bereiche betreffen, die bereits auf Schemaebene geändert wurden. Damit wird auch das Risiko reduziert, dass später bei der Migration Konflikte auftreten. Ganz verhindert werden kann dies nur, wenn anders herum auch die Instanz-Modifikationen zu Sperren bei der Schemaänderung führen. Allerdings würde das die Vorlagenänderung einschränken und sollte daher nicht angewendet werden.

Abbildung 5.8 zeigt, wie durch ein Rückgreifen auf die während der Vorlagenänderung gesetzten Sperren die in Abbildung 5.7 gezeigte Situation vermieden werden kann.

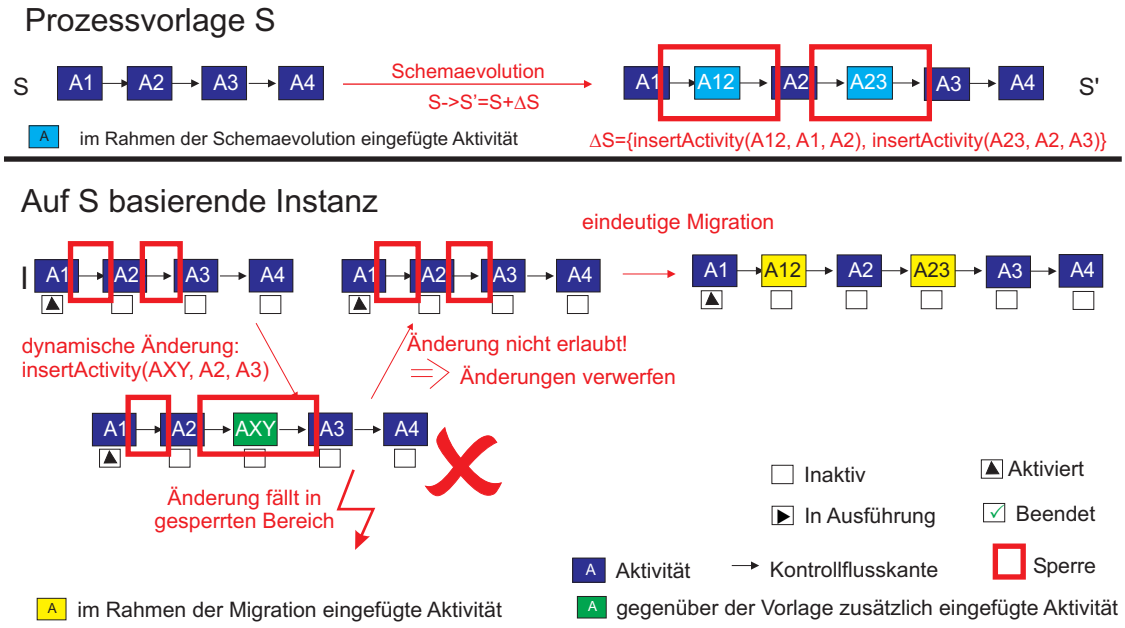

Abbildung 5.8: Reduzierung des Überlappungsgrades von Instanz- und Schemaänderungen

Infolge der Einfügung der Aktivität A12 zwischen die Aktivitäten A1 und A2 auf Typebene ist der Bereich zwischen A1 und A2 gesperrt worden, damit andere Benutzer, welche dieselbe Vorlage abändern, diesen Bereich nicht auch modifizieren und damit u. U. Konflikte hervorrufen

können. Diese Sperre verhindert dann aber auch, dass auf Instanzebene an gleicher Stelle die Aktivität AXY eingefügt werden kann und dass es somit bei der Migration dieser Instanz zu dem in Abbildung 5.7 illustrierten Problem kommt, sich zwischen mehreren denkbaren Anpassungen der Instanz an die Vorlage entscheiden zu müssen.

5.4 Dynamische Änderungen ↔ Weiterschalten

Das parallele Weiterschalten und dynamische Abändern von Instanzen ist ein weiterer Fall von konkurrierenden Zugriffen, bei dem untersucht werden muss, ob sich daraus Probleme ergeben können.

Um eine Instanz abzuändern, wird das Instanz-Objekt, das typbestimmende Vorlagen-Objekt und die eventuell bereits vorhandenen Delta-Schicht-Objekte auf den Clientrechner kopiert. Die Änderungen werden dann entweder auf einem der kopierten Delta-Schicht-Objekte ausgeführt oder es wird ein neues angelegt. Beim Commit wird das entsprechende Delta-Schicht-Objekt an den Server zurückgeschickt und in die Originalinstanz integriert. Da die Modifikationen zunächst auf Kopien durchgeführt werden, kann das Weiterschalten der Instanz ungestört und ohne Probleme gemäß dem durch das bisherige Laufzeitschema vorgeschriebene Vorgehen erfolgen.

Allerdings kann es in Bezug auf die Verträglichkeit der Änderungsoperationen mit der Instanz zu Problemen kommen: Vor der Ausführung einer jeden Operation muss deren Verträglichkeit mit dem eingenommenen Laufzeitzustand der Instanz überprüft werden. Nur wenn die für die entsprechende Operation relevanten Aktivitäten die richtigen, für die Verträglichkeit der Operation vorausgesetzten Zustände aufweisen, darf die Modifikation durchgeführt werden. Da aber die Modifikationen zunächst auf Instanzkopien ausgeführt werden, kann zwischen der Verträglichkeitsprüfung und dem endgültigen Einspielen der Änderungen auf die Originalinstanz im Rahmen der Commit-Behandlung eine längere Zeit vergehen. Ohne besondere Vorkehrungen ist es dann möglich, dass in dieser Zeitspanne die bereits überprüften Aktivitäten im Nachhinein in Zustände geschaltet werden, welche die Anwendung der Operationen beim Commit nicht mehr erlauben dürften. Werden die Operationen aber trotzdem, ohne eine erneute, abschließende Verträglichkeitsprüfung übernommen, dann kann es zu Inkonsistenzen im Zustand der Instanz kommen, wie das Beispiel aus Abbildung 5.9 zeigt.

Für die Ad-hoc-Modifikation der Instanz I wird diese auf den Client kopiert. Um in die Instanz die Aktivität A12 zwischen A1 und A2 einfügen zu können, darf A2 sich höchstens im Zustand „AKTIVIERT“ befinden. Die auf dem Client stattfindende Verträglichkeitsprüfung zum Zeitpunkt t1 für die Operation $insertActivity(A12,\ A1,\ A2)$ ergibt, dass I verträglich mit der gewünschten Operation ist, da die Bedingung an den Zustand von A2 erfüllt ist: A2 ist nur aktiviert. Auf den positiven Ausgang der Verträglichkeitsprüfung hin wird die Aktivität bei t2 auf die Instanzkopie eingefügt. Bevor aber das Einspielen dieser Operation auf den Server vorgenommen wird, wird die Aktivität A2 dort gestartet und I damit in einen mit der Einfüge-Operation unverträglichen Zustand gebracht. Erfolgt das Einspielen dann ungeprüft, kommt

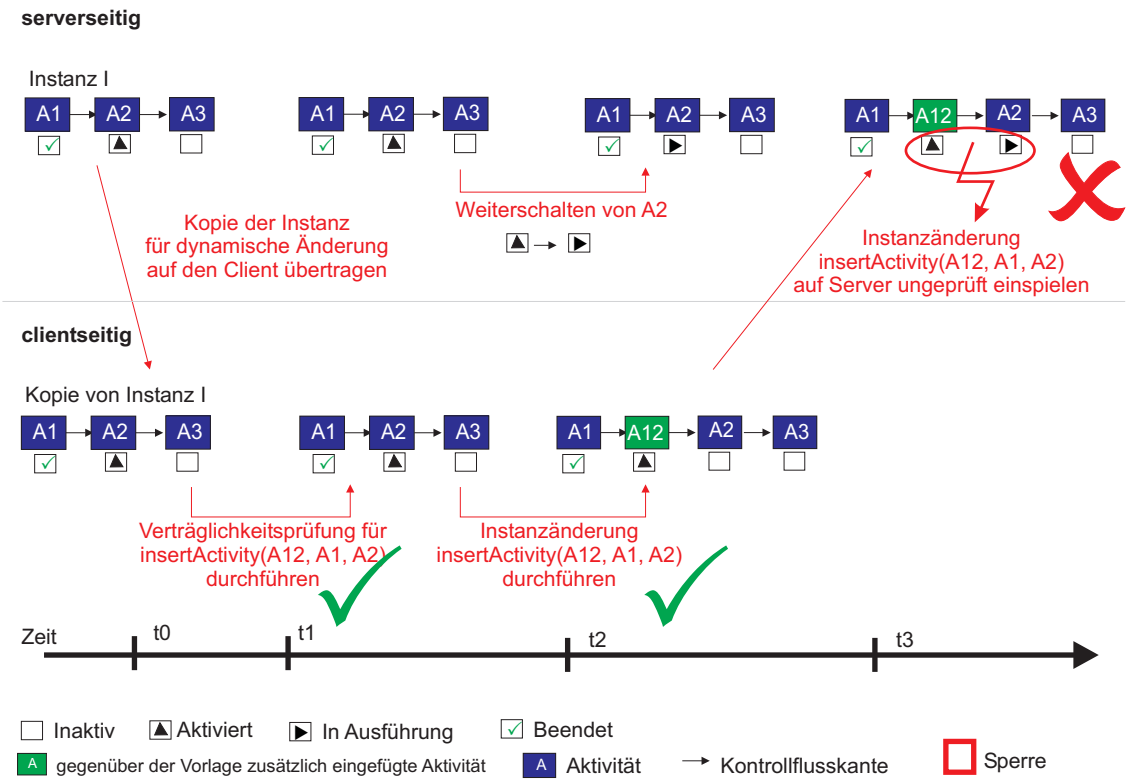

Abbildung 5.9: Konflikt bei konkurrierender Instanzänderung und Aktivitäten-Weiterschaltung

es zu der Zustandsinkonsistenz, dass A2 gestartet wurde, bevor ihre Vorgängeraktivität A12 ausgeführt wurde.

Es muss somit verhindert werden, dass es aufgrund eines zu den dynamischen Änderungen parallel erfolgten Weiterschaltens einer Instanz zu Inkonsistenzen beim Einspielen der Änderungen kommt.

Zwei Ansätze können zu diesem Ziel führen: Zum einen kann, wie in der Problembeschreibung bereits angedeutet, eine erneute Verträglichkeitsprüfung im Rahmen der Commit-Behandlung für eine korrekte Instanzänderung sorgen. Dies bringt aber den Nachteil mit sich, dass entweder der Benutzer bei einem Scheitern der Verträglichkeitsprüfung seine Instanzanpassungen überarbeiten muss oder dass einige Aktivitäten wieder zurückgesetzt werden müssen. Das Zurücksetzten ist aber im Allgemeinen aufwändig und nicht immer möglich.

Ein anderer Weg besteht darin zu verhindern, dass sich der Zustand einer bereits überprüften Aktivität bis nach dem endgültigen Einspielen der Operationen noch einmal ändert. Dies kann analog wie bei dem Fall „Vorlagenänderung ↔ Weiterschalten“ (siehe Abschnitt 5.2) erreicht werden, indem die Instanz während der Ad-hoc-Modifikation komplett gegen ein Weiterschalten

gesperrt wird oder indem (nur) die bereits überprüften Aktivitäten der Instanz dagegen gesperrt werden.

Die Nachteile der beiden Varianten sind dieselben, wie in Abschnitt 5.2 diskutiert: Die Instanz komplett zu sperren blockiert auch Aktivitäten, die nicht von den Verträglichkeitsprüfungen angefasst werden. Nur die Aktivitäten an der Ausführung zu hindern, die für die bisher angewendeten Änderungsoperationen relevant sind, bringt den Nachteil mit sich, dass mit jedem Weiterschalten einer nicht gesperrten Aktivität die Menge an prinzipiell noch vornehmbaren Modifikationen des Laufzeitschemas reduziert wird, d. h. jedes Weiterschalten verringert die Chance, dass der Benutzer seine gewünschten Änderungen durchbringen kann.

Manche Aktivitäten nicht gegen das Weiterschalten zu sperren hat auch den Nachteil, dass die Instanzkopien auf den Clientrechnern und die jeweiligen Originale vom Zustand her ständig synchron gehalten werden müssen, d. h. jede Kopie muss über einen erfolgten Zustandswechsel zuverlässig informiert werden.

Alle drei Konzepte – das erneute Prüfen beim Einspielen, die Komplettsperre und das Sperren der relevanten Aktivitäten – führen zu dem Ziel, dass es zu keinen Inkonsistenzen beim Einspielen der Änderungen kommt. Allerdings weisen alle drei Nachteile auf. Wiederum muss man abwägen, welches Verfahren angewendet werden soll. Da dynamische Änderungen von Instanzen eher von geringem Umfang sind und damit im Allgemeinen relativ schnell erfolgen, bietet sich die Lösung mit der Komplettsperre an. Sie erfordert wenig Verwaltungsaufwand für das System.

5.5 Dynamische Änderungen ↔ Dynamische Änderungen

Wie auch mehrere Benutzer parallel die gleiche Vorlage anpassen wollen können, so können auch mehrere Benutzer zeitgleich versuchen, ein und dieselbe Instanz dynamisch abzuändern.

Daraus können dieselbe Probleme entstehen, wie im Fall „Vorlagenänderung ↔ Vorlagenänderung“ beschrieben (siehe Abschnitt 5.1): Zum Beispiel können unterschiedliche Aktivitäten an derselben Stelle eingefügt werden oder das aus den verschiedenen Änderungstransaktionen resultierende Schema kann gegen die Korrektheitskriterien des zugrunde gelegten Prozess-Modells verstoßen.

Analog zum Fall „Vorlagenänderung ↔ Vorlagenänderung“ besteht das zu erreichende Ziel darin, dass Konflikte aufgrund von überlappenden Änderungen vermieden oder aufgelöst werden.

Da die gleichen Probleme wie im Fall „Vorlagenänderung ↔ Vorlagenänderung“ auftreten, können auch dieselben Verfahren zur Vermeidung bzw. Auflösung von Konflikten aufgrund von überlappenden Änderungen herangezogen werden:

- Exklusivzugriff
- Bestätigung einer Änderung durch den Server

- Sperren
- Optimistisches Verfahren

Diese Verfahren wurden bereits in Abschnitt 5.1 detailliert beschrieben.

5.6 Migration ↔ Weiterschalten

Bisher wurde die Problematik von konkurrierenden Benutzeraktionen untersucht. Aber es kann auch zu Konflikten oder anderweitigen Wechselwirkungen zwischen Benutzeraktionen und der Migration kommen, die zu den Systemfunktionen zählt.

Unter anderem muss die Wechselwirkung „Migration ↔ Weiterschalten“ näher betrachtet werden.

Das Weiterschalten einer Instanz kann mit einer Migration konkurrieren, wenn genau die Instanz weitergeschaltet wird, die gerade im Begriff ist, von der alten Vorlagenversion S auf die neue S' zu migrieren. Das Weiterschalten der Instanz kann ohne weitere Absicherungen durch das System mehrere Konflikte verursachen.

Zu untersuchen ist, welche Konflikte aus dem Zusammenspiel zwischen der Migration und dem zeitgleichen Weiterschalten einer Instanz entstehen können und wie sie zu vermeiden sind.

Um eine Instanz auf die neue Version S' der Vorlage migrieren zu können, muss sie verträglich mit den Vorlagenänderungen sein. Dafür dürfen die für die Verträglichkeit relevanten Aktivitäten nur bestimmte Zustände einnehmen. In der Phase der Verträglichkeitsprüfung des Migrationsprozesses wird die Einhaltung dieser Bedingungen überprüft. Wird die Einhaltung bestätigt, dann kann die Instanz mit der Migration fortfahren, d. h. sie ist verträglich. Ansonsten ist die Instanz unverträglich und wird von der Migration ausgeschlossen.

Im Allgemeinen sind mehrere Aktivitäten zu überprüfen. Wird eine dieser Aktivitäten in dieser Phase weitergeschaltet, noch bevor sie zur Überprüfung an der Reihe war, so kann dies dazu führen, dass sie in einen Zustand gelangt, mit dem nicht mehr die Bedingung an deren Zustand für eine Verträglichkeit erfüllt ist. Der Test wird dann spätestens bei dieser Aktivität die Instanz korrekt für unverträglich erklären und sie damit von der Migration ausschließen, auch wenn die Instanz zu Beginn der Phase der Verträglichkeitsprüfung vom Zustand her verträglich mit den Änderungen war.

Folgenreicher aber ist der Fall, dass die Aktivität noch nach ihrer positiven Überprüfung, aber noch vor dem Ende dieser Phase weitergeschaltet wird. Gelangt die Aktivität dabei in einen Zustand, der den Bedingungen widerspricht, so kann das, wie Abbildung 5.10 zeigt, dazu führen, dass der Test die Instanz am Schluss fälschlicherweise für verträglich erklärt, obwohl die Instanz eigentlich aufgrund des neuen Zustandes dieser Aktivität unverträglich ist.

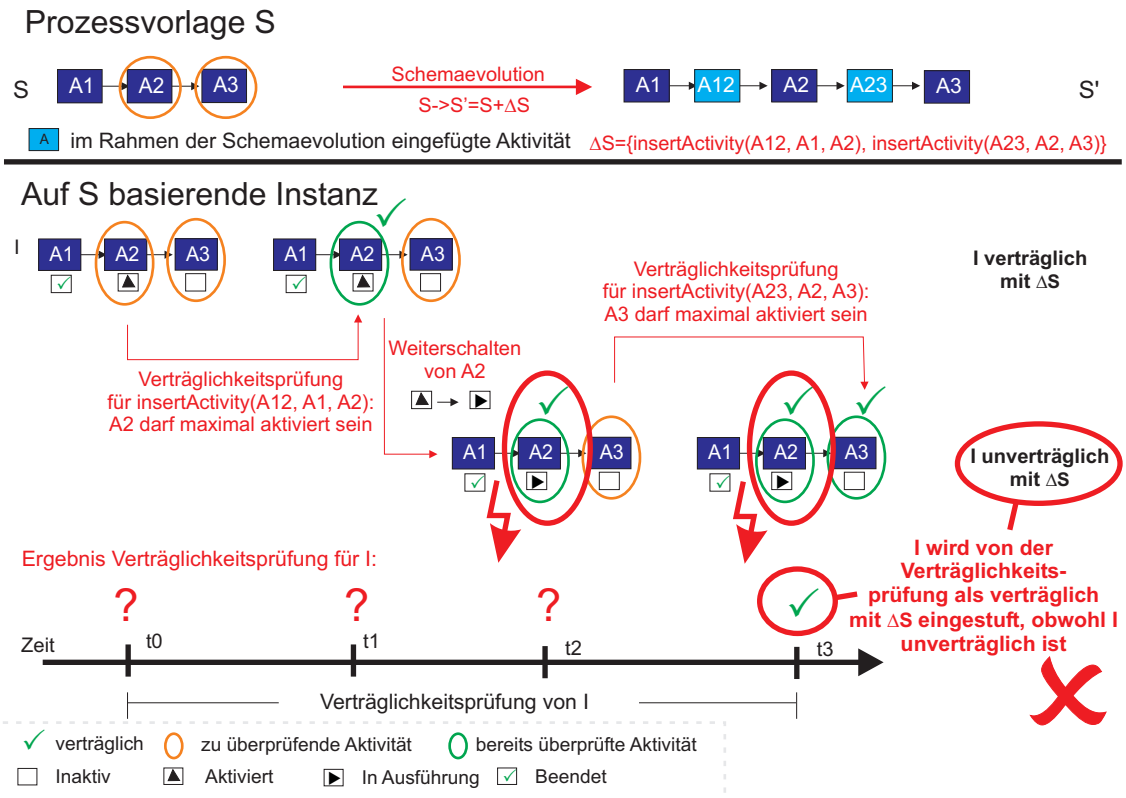

Abbildung 5.10: Inkorrekte Verträglichkeitsprüfung aufgrund einer Zustandsweiterschaltung

Wird die Instanz daraufhin in der vierten Phase auf das neue, S' repräsentierende Vorlagen-Objekt umgehängt, dann werden – logisch gesehen – Änderungen des Laufzeitschemas der Instanz durchgeführt, die aufgrund des Zustandes der Instanz nicht erfolgen dürften. Zustandsinkonsistenzen und Fehler bei der Prozessausführung drohen im folgenden.

Derselbe Umstand ergibt sich, wenn einer der überprüften Aktivitäten im Zeitraum zwischen dem korrekten Ende der Verträglichkeitsprüfung und dem Umhängen der Instanz auf das neue Vorlagen-Objekt durch das Weiterschalten in einen verbotenen Zustand gelangt.

Deshalb muss mit dem Start der Verträglichkeitsprüfung sichergestellt werden, dass keine der relevanten Aktivitäten ihren Zustand mehr ändern kann, bis entweder der Test die Instanz aufgrund ihrer Unverträglichkeit von der Migration ausgeschlossen hat oder bis die Instanz im Falle der Verträglichkeit vollständig die vierte Phase des Migrationsprozesses durchlaufen hat und das entsprechende Instanz-Objekt korrekt auf das neue Vorlagen-Objekt umgehängt wurde.

Eine Möglichkeit, dies sicherzustellen und damit Konflikte zu vermeiden, ist, die für die Verträglichkeitsprüfung relevanten Aktivitäten einer Instanz zu Beginn des Tests bis zum Abschluss

der vierten Phase des Migrationsprozessen gegen das Weiterschalten zu sperren, wie Abbildung 5.11 zeigt:

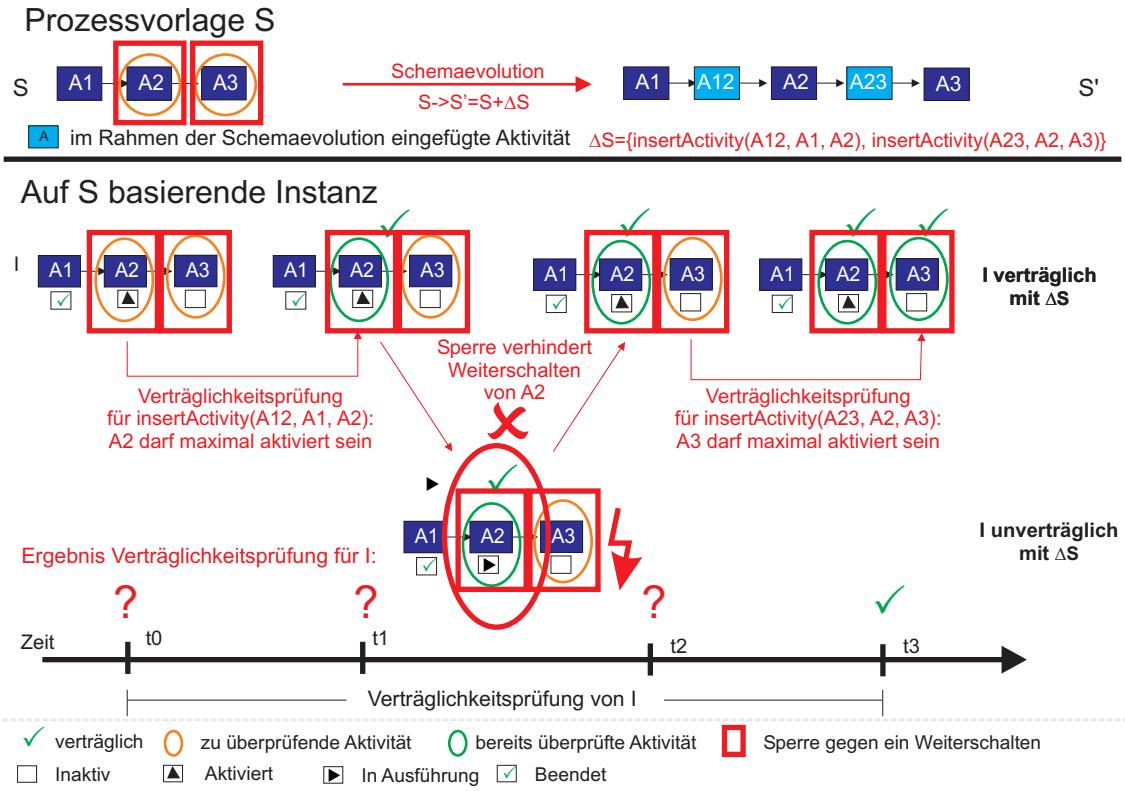

Abbildung 5.11: Verhinderung inkorrekter Verträglichkeitsprüfungen durch Sperren

Das Sperren der für die Verträglichkeitsprüfung relevanten Aktivitäten gegen das Weiterschalten verhindert bei der Instanz I aus Abbildung 5.11, dass die Aktivität A2, die zur Zeit t1 ihrer Überprüfung einen für die Verträglichkeit zulässigen Zustand inne hat, nach ihrer Überprüfung aber noch vor dem Ende der Verträglichkeitsprüfung in einen für die Verträglichkeit der Instanz unzulässigen Zustand geschalten werden kann und dass damit das Ergebnis der Verträglichkeitsprüfung verfälscht wird, wie in Abbildung 5.10 demonstriert worden ist.

Der Vorteil dieser Methode gegenüber einer Komplettsperre der Instanz ist, dass die nicht relevanten Aktivitäten ausgeführt werden können und somit nicht unnötig behindert werden. Der Nachteil ist, dass das Sperren einzelner Aktivitäten einen höheren Verwaltungsaufwand verursacht als eine Komplettsperre.

Wie in der Untersuchung des Falls „Vorlagenänderung ↔ Weiterschalten“ in Abschnitt 5.2 diskutiert, kann es sinnvoll sein, das Weiterschalten der relevanten Aktivitäten schon während der Vorlagenänderung selbst zu unterbinden und nicht erst nach dem Beginn der Verträglichkeitsprüfung. Greifen dann aber noch viele andere Instanzen des gleichen Prozesstyps auf diese

Sperrtabelle zu, um präventiv ein Weiterschalten bis zu deren Migration zu unterbinden, dann kann diese Datenstruktur zum Flaschenhals werden.

Das Sperren der für die Verträglichkeitsprüfung relevanten Aktivitäten gegen ein Weiterschalten stellt nicht nur sicher, dass eine als verträglich eingestufte Instanz auch bis zum Umhängen auf die neue Vorlage verträglich bleibt. Es verhindert auch, dass die Instanz in dem Zeitraum zwischen dem positiven Ausgang der Verträglichkeitsprüfung und dem Umhängen auf die neue Vorlage S' nach der alten Version S abläuft anstatt nach der neuen Version S' und damit Änderungen auslässt, die das Laufzeitschema der Instanz erst nach dem Umhängen auf die neue Vorlage aufweisen wird.

Aber auch nach dem Umhängen des diese Instanz repräsentierenden Instanz-Objekts auf das neue Vorlagen-Objekt können Konflikte aufgrund eines Weiterschaltens der Instanz vor dem Abschluss ihrer Migration auftreten.

Wird eine Aktivität beendet, so werden gemäß Vorgabe des Schemas die als nächstes auszuführenden Aktivitäten bestimmt, deren Status auf „AKTIVIERT“ gesetzt und dann entsprechende Arbeitsaufträge in die Arbeitslisten der infrage kommenden Bearbeiter gestellt. Um aber den Status einer Aktivität auf „AKTIVIERT“ setzen zu können, muss für diese Aktivität in der Datenstruktur, welche die Laufzeitzustände der verschiedenen Aktivitäten einer Instanz speichert, ein entsprechender Eintrag vorhanden sein. Dieser Eintrag wird für Aktivitäten, die im Rahmen der Schemaevolution eingefügt worden sind, in der fünften Phase des Migrationsprozesses, der Phase der Instanzanpassung, angelegt.

Findet dieser Zustandswechsel nach der vierten Phase des Migrationsprozesses – die Instanz läuft damit bereits nach dem neuen Schema ab – aber noch vor Ende der Instanzanpassung statt und zählt die Aktivität zu den neu hinzugefügten, dann besteht die Möglichkeit, dass für diese Aktivität noch kein Eintrag in dieser Datenstruktur angelegt wurde. Der Versuch, diese Aktivität zu aktivieren, kann dann u. U. zu einem undefinierten Verhalten des Systems führen.

Eine ähnliche Situation liegt vor, wenn die neue Aktivität in einem XOR-Zweig liegt und dieser abgewählt wird. Existiert für die entsprechende Aktivität noch kein Eintrag in der angesprochenen Datenstruktur, dann kann der Status der Aktivität auch nicht auf „ABGEWÄHLT“ gesetzt werden. Auch in diesem Fall droht u. U. ein undefiniertes Verhalten des Systems.

Somit muss verhindert werden, dass sich der Status neu hinzugefügter Aktivitäten vor dem Ende der 5. Phase des Migrationsprozesses ändern kann. Wiederum können dazu Sperren eingesetzt werden.

Zu weiteren Konflikten kann es kommen, wenn aktivierte Aktivitäten, die im Rahmen der Markierungsanpassung wieder deaktiviert werden müssen, zuvor gestartet werden. Wird z. B. vor die aktivierte Aktivität Z die Aktivität Y eingefügt, dann muss, wie in Kapitel 2.3.4 erklärt, Z im Rahmen der Markierungsanpassung in den inaktiven Zustand zurückversetzt werden und dafür Y aktiviert werden. Wird jedoch Z vor dem Zurücksetzen gestartet, dann wird Z ausgeführt, obwohl dazu nach dem neuen Schema zuvor Y beendet worden sein hätte müssen. Es kommt zu einer Zustands- bzw. Ausführungsinkonsistenz.

Die Ausführung der Aktivität Z rückgängig zu machen, kann das Problem lösen. Viele Aktivitäten können aber in der Praxis nicht mehr so einfach rückgängig gemacht werden. Deshalb scheidet diese Lösung oft aus. Die für die Markierungsanpassung relevanten Aktivitäten gegen das Weiterschalten zu sperren bietet Abhilfe. Die Menge der zu sperrenden Aktivitäten kann wiederum anhand der ausgeführten Änderungsoperationen ermittelt werden.

Probleme zwischen einem Weiterschalten und der zweiten und dritten Phase des Migrationsprozesses treten nicht auf, da die Klasseneinteilung und die Migrationsvorbereitung ausschließlich mit den statischen Schemainformationen der alten und neuen Vorlage arbeiten und den dynamischen Aspekt einer Instanz außer Acht lassen.

Eine Instanz weiterzuschalten und gleichzeitig zu migrieren funktioniert somit nur eingeschränkt: Einige Aktivitäten müssen gegen das Weiterschalten gesperrt werden, um eine korrekte Migration zu garantieren.

Wenn aber die Dauer der Migration einer Instanz kurz oder ungefähr gleich lang verglichen mit der durchschnittlichen Zeit ist, die zwischen zwei Zustandsänderungen derselben Instanz aufgrund von Weiterschalten vergeht, dann ist eine sinnvolle Alternative, die Instanz während der gesamten Migration komplett gegen das Weiterschalten zu sperren, anstatt nur einige Aktivitäten.

Für die Entscheidung über das Verfahren muss aber auch die Verwaltung der Arbeitslisten berücksichtigt werden: Ist es effizienter, nur gezielt einige Arbeitsaufträge aus den Listen zu entfernen und später auch nur wieder einzelne Einträge dort hineinzustellen, oder ist es effizienter, alle Einträge zu einer Instanz zu Beginn der Migration herauszunehmen und am Schluss wieder alle zu einer Instanz gehörigen Einträge auf einmal einzustellen?

5.7 Migration ↔ Vorlagenänderung

Die Migration kann nicht nur mit dem Weiterschalten einer Instanz konkurrieren, sondern auch mit einer Vorlagenänderung.

Dieser Fall liegt vor, wenn eine Vorlage $S' = S + \Delta S$ im Rahmen einer Schemaevolution durch die Modifikationen $\Delta S'$ zu $S'' = S' + \Delta S'$ abgeändert wird, gleichzeitig aber noch Instanzen von der Vorgängerversion S auf diese Version S' migrieren.

Diesbezüglich ist zu klären, ob es dabei zu Problemen kommt und was beachtet werden muss.

Dafür müssen die verschiedenen Phasen der Migration getrennt auf Wechselwirkungen mit den Vorlagenänderungen untersucht werden.

Verträglichkeitsprüfung ↔ Vorlagenänderung

Für die Verträglichkeitsprüfung einer Instanz werden die aktuellen Ausführungszustände bestimmter, von den angewendeten Operationen abhängiger Aktivitäten benötigt und eventuell Informationen aus der Ausführungshistorie (wenn Sync-Kanten eingefügt werden) sowie aus der Datenhistorie (wenn z. B. Datenelemente gelöscht werden) (vgl. [RRD02]). Da die Vorlagenänderung weder den aktuellen Ausführungszustand einer Instanz noch deren Ausführungs- oder Datenhistorie beeinflusst, kommt es zu keinen Wechselwirkungen zwischen der Vorlagenänderung S' nach S'' und der Verträglichkeitsprüfung der Instanz bezüglich S'.

Zur Bestimmung der Menge der für die Verträglichkeitsprüfung zu untersuchenden Aktivitäten werden Schemainformationen der Version S benötigt, nach der die Instanz bisher abgelaufen ist. Da durch die Schemaänderung $\Delta S'$ nur die Version S' der Vorlage abgeändert wird, auf welche die Instanz zu migrieren ist, kommt es auch hierbei zu keinen Problemen.

Somit sind für diese Phase der Migration keine weiteren Maßnahmen notwendig.

Klasseneinteilung ↔ Vorlagenänderung

Um die Überlappung der Instanzänderungen einer zu migrierenden, verzerrten Instanz mit den Schemaänderungen $\Delta S'$ zu bestimmen, werden u. a. die Schemainformationen der Version S' benötigt (vgl. [Ri04]). Da aber die Modifikationen $\Delta S'$ auf einer Kopie des Vorlagen-Objekts, das die Version S' repräsentiert, ausgeführt werden, die Schemainformationen aber auf dem originalen Vorlagen-Objekt abgegriffen werden, besteht auch in dieser Phase der Migration keine Gefahr der Wechselwirkung mit den parallel dazu vorgenommenen Schemaänderungen. Wiederum sind keine weiteren Maßnahmen zu treffen.

Migrationsvorbereitung ↔ Vorlagenänderung

In der Phase der Migrationsvorbereitung wird bei einer mit dem Bias ΔS_I gegenüber S verzerrten Instanz u. a. überprüft, ob deren Migration auf die neue Vorlagenversion S' zu einem inkorrekten Laufzeitschema S_I*' der Instanz führen wird, z. B. weil die auf Instanz- und die auf Typebene eingefügten Sync-Kanten nach der Migration zu einem Zyklus im Kontrollfluss führen. Dafür werden neben den auf Instanz- und Typebene angewandten Änderungsoperationen ΔS_I und ΔS die Schemainformationen der ursprünglichen Vorlagenversion S benötigt (vgl. [Ri04]). Da mit der jetzigen Schemaevolution $S' \rightarrow S''$ keine dieser Informationen geändert werden, kommt es in dieser Beziehung auch in dieser Phase nicht zu Problemen zwischen der parallel ablaufenden Migration und den Vorlagenänderungen.

Zusätzlich wird in dieser Phase auch der neue Bias $\Delta S'_I$ berechnet, mit dem die bisher mit dem Bias ΔS_I gegenüber S verzerrte Instanz nach der Migration gegenüber S' verzerrt sein wird, so dass für ihr neues Laufzeitschema S_I*' gilt: $S_I*' \equiv_{trace} S' + \Delta S'_I$. Zur Berechnung von $\Delta S'_I$ werden neben den auf Instanz- und Typebene vorgenommenen Änderungen ΔS_I und

ΔS das Laufzeitschema $S_I* \equiv_{trace} S + \Delta S_I$ der Instanz vor der Migration und die beiden Vorlagenversionen S und S' benötigt (vgl. [Ri04]). Die Schemaevolution $S' \rightarrow S''$ beeinflusst weder die gemachten Änderungen ΔS_I und ΔS noch das Laufzeitschema S_I* der Instanz vor der Migration noch die ursprüngliche Version S der Prozessvorlage. Aber auch die Vorlage S' bleibt im System vollständig erhalten, da die im Rahmen der Vorlagenänderung vorgenommenen Schemaanpassungen $\Delta S'$ auf einer Kopie des Vorlagen-Objekts, das S' repräsentiert, ausgeführt werden und nicht auf dem Original selbst.

Damit stören bzw. beeinflussen die Vorlagenänderungen auch nicht diese Phase der Migration.

Migrationsvorgang ↔ Vorlagenänderung

In dieser Phase des Migrationsprozesses wird eine verträgliche Instanz auf das Vorlagen-Objekt, das die neue Vorlagenversion S' repräsentiert, umgehängt und – wenn nötig – die Delta-Schicht eingefügt, die für den in der vorangegangenen Migrationsphase berechneten, neuen Bias $\Delta S'_I$ sorgt. Wie bereits in den Untersuchungen zur vorigen Phase ausgeführt, bleibt das Vorlagen-Objekt unangetastet von den erneuten Vorlagenänderungen $\Delta S'$, da diese auf einer Kopie dieses Objekts ausgeführt werden. Somit kommt es auch hier zu keinen Problemen aufgrund der parallelen Vorgänge.

Instanzanpassung ↔ Vorlagenänderung

Auch die Instanzanpassung wird nicht durch die Vorlagenänderung beeinflusst. Bei der Instanzanpassung werden z. B. die Datenstrukturen, welche sich die Zustände der einzelnen Aktivitäten merken, um Einträge für die durch die Migration neu hinzugekommenen Aktivitäten erweitert oder um die für gelöschte Aktivitäten reduziert. Die Menge der eingefügten und gelöschten Aktivitäten kann allein aus den Betrachtungen der Instanz- und Schemaänderungen ΔS_I und ΔS gewonnen werden. Und die neue Vorlagenänderung hat keine Auswirkungen darauf.

Markierungsanpassung ↔ Vorlagenänderung

Ausschließlich in der letzten Phase des Migrationsprozesses, der Markierungsanpassung, kann es zu Wechselwirkungen zwischen der Migration und der Vorlagenänderung kommen.

Bei der Markierungsanpassung werden bestimmte, von den angewandten Änderungsoperationen abhängige, bereits aktivierte Aktivitäten in den inaktiven Zustand zurückgesetzt, andere dagegen in den „AKTIVIERT“-Modus versetzt, der signalisiert, dass diese bereit zur Ausführung sind. Die Folge ist, dass die infrage kommenden Bearbeiter entsprechende Einträge in ihre Arbeitslisten gestellt bekommen.

Um eine korrekte Markierungsanpassung durchführen zu können, werden neben dem aktuellen Status der Instanz die Informationen über die auf Instanz- und Typebene vorgenommenen

Änderungen ΔS_I und ΔS benötigt, sowie die Schemainformationen alter und neuer Version S bzw. S'. Da die Schemaänderungen $\Delta S'$ auf einer Kopie von S' ausgeführt werden und die anderen Informationen auch nicht von der neuen Schemaevolution beeinflusst werden, kommt es zu keinen Problemen.

Allerdings wurde in Kapitel 5.2 ausgeführt, dass das Weiterschalten eine Instanz in einen unverträglichen Zustand mit den parallel dazu durchgeführten Änderungen ihrer Vorlage bringen kann und dass dies durch Sperren der ganzen Instanz oder der unmittelbar betroffenen Aktivitäten ganz oder zumindest teilweise verhindert werden kann.

Werden nun während der Vorlagenänderung S' nach S'' die auf S' basierenden Instanzen oder wenigstens die von den Änderungen $\Delta S'$ betroffenen Aktivitäten gegen ein Weiterschalten gesperrt, dann muss dies bei der Markierungsanpassung einer Instanz beachtet werden, die von der Version S auf S' migriert. Es muss verhindert werden, dass Arbeitsaufträge zu den neu aktivierten Aktivitäten in die Arbeitslisten der zugeordneten Bearbeiter gestellt werden, wenn die entsprechenden Aktivitäten als gesperrt markiert worden sind.

Zusammengefasst kann gesagt werden, dass nichts gegen die Migration einer Instanz auf eine Version der Vorlage spricht, die zeitgleich erneut abgeändert wird. Einzig und allein die Sperren, die gesetzt werden, um eine Unverträglichkeit aufgrund von Weiterschalten zu verhindern, müssen bei der Markierungsanpassung beachtet werden.

5.8 Migration ↔ Dynamische Änderungen

Komplizierter ist die Situation zwischen den konkurrierenden Aktionen Migration und dynamische Änderung der zu migrierenden Instanz.

In dieser Situation wird versucht, eine eventuell schon verzerrte Instanz I, die gerade im Begriff ist, von der Vorlagenversion S nach S' zu migrieren, dynamisch (erneut) in ihrem Laufzeitschema zu ändern. Da sowohl die Migration als auch die dynamische Änderung Modifikationen an dem Laufzeitschema der Instanz und an dem diese Instanz repräsentierenden Instanz-Objekt vornehmen und zusätzlich beide den Zustand der Instanz anpassen, sind Konflikte zu erwarten.

Dieser Abschnitt klärt, wann Konflikte auftreten und was getan werden muss, um diese zu verhindern.

In der ersten Phase der Migration, der Verträglichkeitsprüfung, kommt es zu keinen Konflikten, wenn die Instanz parallel dazu abgeändert wird: Keine Instanzänderung führt dazu, dass eine bisher noch nicht ausgeführte Aktivität in den Zustand „IN AUSFÜHRUNG“ gelangt. Da für die Unverträglichkeit einer Instanz mit den Schemaänderungen aber ausschließlich Aktivitäten verantwortlich sind, die sich mindestens im Zustand „IN AUSFÜHRUNG“ befinden, kann sich der Verträglichkeitsstatus einer Instanz durch deren Ad-hoc-Modifikation nicht ändern.

Auch kommt es zu keinen Konflikten, wenn im Rahmen der Verträglichkeitsprüfung der Zustand einer Aktivität überprüft werden muss, die im Rahmen der Instanzänderung zuvor her-

ausgelöscht worden ist. Der Zustand dieser Aktivität kann automatisch als verträglich angenommen werden, denn diese Aktivität konnte vor dem Löschen maximal aktiviert gewesen sein. Ansonsten hätte sie nicht im Rahmen der Ad-hoc-Modifikation gelöscht werden können.

Erst bei der Klasseneinteilung, der zweiten Phase des Migrationsprozesses, können Probleme auftreten.

In dieser Phase des Migrationsprozesses wird bestimmt, inwieweit sich die dynamischen Änderungen der zu migrierenden Instanz mit den Schemaäenderungen überlappen. Für die Bestimmung des Überlappungsgrades muss ein stabiles Laufzeitschema der Instanz vorliegen, ansonsten kommt es zu einem inkorrekten Ergebnis, wie im folgenden veranschaulicht wird.

Für die Bestimmung ist neben vielen weiteren Informationen das Ergebnis eines Vergleichs zwischen der Menge der auf Instanzebene eingefügten Aktivitäten und der Menge der auf Typebene eingefügten Aktivitäten relevant [Ri04][1]. Die Aktivitäten, die im Rahmen einer zu der Migration parallel erfolgenden Instanzänderung eingefügt werden, müssen bei diesem Vergleich mit einbezogen werden. Wird aber eine Aktivität erst eingefügt, nachdem dieser Vergleich bereits erfolgt ist, dann spiegelt das Ergebnis des Vergleichs nicht mehr die nun gegebene Situation wider und die Überlappungsbestimmung erfolgt auf Basis veralteter Informationen. Im Allgemeinen wird dann ein Überlappungsgrad ermittelt, der von dem aktuell zwischen Instanzänderung und Schemaänderung vorherrschenden Überlappungsgrad abweicht.

Um also ein korrektes Ergebnis liefern zu können, muss während der Phase der Klasseneinteilung ein stabiles Laufzeitschema der zu migrierenden Instanz vorliegen. Somit darf sich die Instanz in dieser Phase nicht ändern.

Der bestimmte Überlappungsgrad legt fest, wie die Instanz in der vierten Phase an die neue Version ihrer Vorlage angepasst werden muss. Deshalb dürfen bis zu der erfolgten Anpassung keine Änderungen an der Instanz durchgeführt werden, die den Überlappungsgrad ändern und damit das Ergebnis der Klasseneinteilung ad absurdum führen.

Die Migration einer Instanz auf Basis inkorrekt gewordener Informationen bezüglich des Überlappungsgrades führen zu Inkonsistenzen bzw. „verlorenen“ Änderungen, wie Abbildung 5.12 zeigt.

Die abgebildete Instanz I ist zum Zeitpunkt der Klasseneinteilung gegenüber ihrer Vorlage S mit dem Bias $\Delta S_I = \{insertActivity(A23, A2, A3)\}$ verzerrt und weist damit ihr gegenüber die zusätzliche Aktivität A23 auf. Im Rahmen der Schemaevolution, welche die Vorlage S in die neue Version S' überführt, wird diese Aktivität allerdings auch auf Typebene hinzugefügt. Da auf Instanzebene keine weiteren Änderungen vorgenommen worden sind, auf Typebene aber zusätzlich noch die Aktivität A12 eingefügt worden ist, wird im Rahmen der Klasseneinteilung für den Überlappungsgrad zwischen der Instanzänderung ΔS_I und den Schemaänderungen ΔS bestimmt: ΔS_I ist subsumptions-äquivalent zu den Schemaänderungen ΔS. Die richtige Migrationsstrategie zur Anpassung der Instanz an die neue Vorlage lautet hierfür, wie in Kapitel 3.6

[1] In [Ri04] ist detailliert beschrieben, wie der Überlappungsgrad bestimmt wird und welche Informationen dazu notwendig sind.

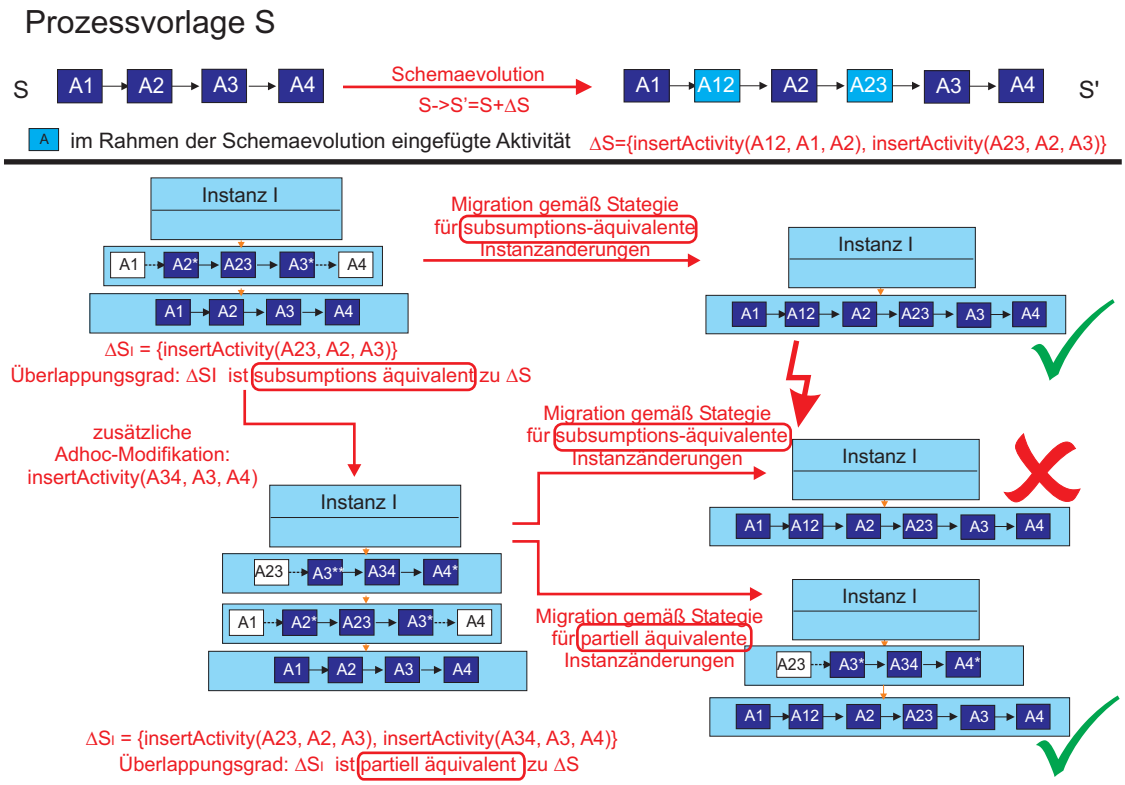

Abbildung 5.12: Inkorrekte Migration aufgrund falscher Informationen über Überlappungsgrad

beschrieben: Das diese Instanz repräsentierende Instanz-Objekt ist einfach direkt auf das neue Vorlagen-Objekt umzuhängen, die existierenden Delta-Schichten sind zu verwerfen. Mit dem Umhängen auf das neue Vorlagen-Objekt wird – logisch gesehen – die zusätzliche Modifikation $insertActivity(A12, A1, A2)$ auf Instanzebene nachgezogen.

Wird nun aber auf Instanzebene noch zusätzlich die Aktivität A34 eingefügt, dann sind die Instanzänderungen nicht mehr subsumptions-äquivalent zu den Schemaänderungen sondern partiell äquivalent, d. h. der Überlappungsgrad hat sich geändert. Wie zuvor muss die Instanz nach der Migration das neue Vorlagen-Objekt (indirekt) referenzieren. Zwischen dem Instanz-Objekt und dem Vorlagen-Objekt muss aber im Falle einer korrekten, dem neuen Überlappungsgrad gemäßen Migration noch eine Delta-Schicht liegen, welche die resultierende Verzerrung $\Delta S_I \setminus \Delta S = \{insertActivity(A34, A3, A4)\}$ der Instanz gegenüber der neuen Schemaversion S' repräsentiert.

Erfolgt die zweite Instanzänderung $insertActivity(A34, A3, A4)$ aber erst nach der Bestimmung des Überlappungsgrades, dann wird der Überlappungsgrad nicht erneut bestimmt, die Instanzänderungen werden fälschlicherweise weiterhin als subsumptions-äquivalent zu den Schemaänderungen betrachtet und die Migration folgt einer falschen Strategie. Die Folge ist, dass

die die Verzerrung $\Delta S_I \setminus \Delta S = \{insertActivity(A34, A3, A4)\}$ repräsentierende Delta-Schicht nicht aufgebaut wird, da die Migrationsstrategie auf der Tatsache beruht, dass Instanzen, bei denen die Instanzänderungen subsumptions-äquivalent zu den Schemaänderungen sind, stets gegenüber der neuen Vorlage unverzerrt sind und deshalb die Bildung einer Delta-Schicht nicht notwendig ist.

Somit muss verhindert werden, dass bis zum Abschluss der vierten Phase Ad-hoc-Änderungen der Instanz vorgenommen werden können.

Wird die Instanz sofort nach dem Umhängen auf das neue Vorlagen-Objekt am Ende der vierten Phase des Migrationsprozesses zur Änderung freigegeben, dann ist folgender Konflikt denkbar:

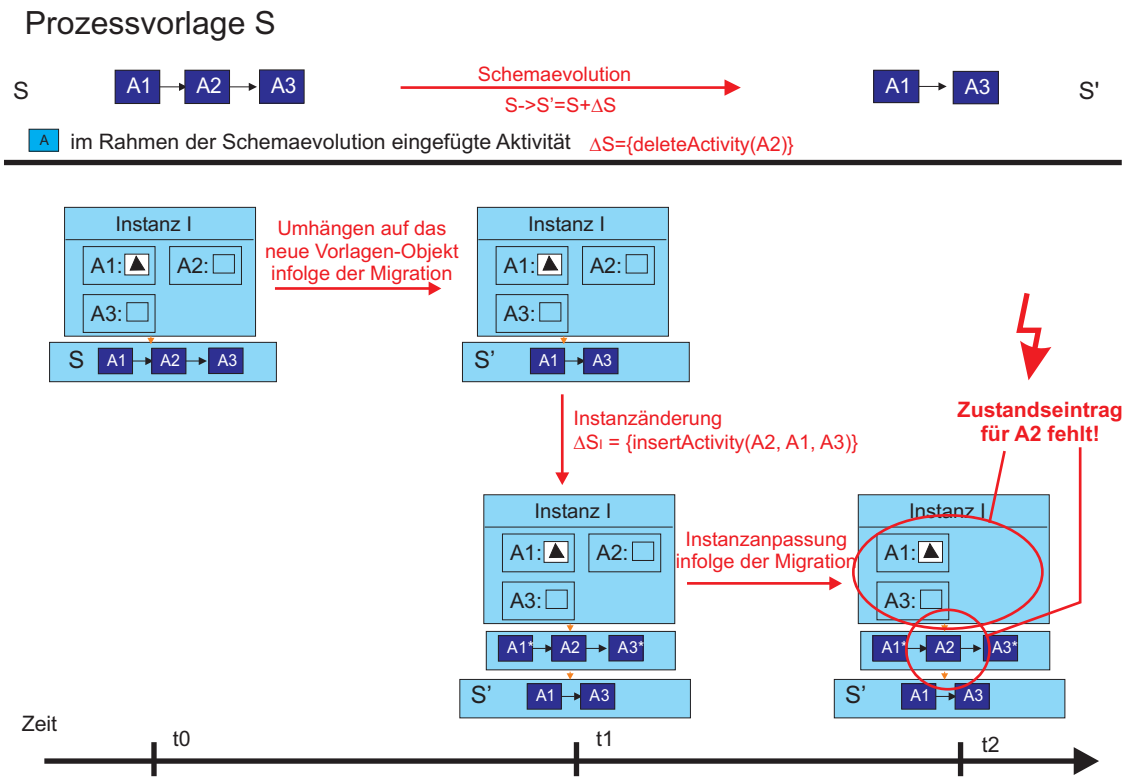

Abbildung 5.13: Scheinbarer Konflikt bei parallel erfolgender Instanzanpassung und -änderung

Das Löschen der Aktivität A2 aus der in Abbildung 5.13 gezeigten Instanz I hat zur Folge, dass alle Einträge im Rahmen der Instanzanpassung bezüglich A2 aus dem Instanz-Objekt herausgelöscht werden. Wird aber noch vor dem Löschen der Einträge dieselbe Aktivität durch eine Instanzänderung wieder eingefügt, dann führt das anschließende Löschen der Einträge im Rahmen der Instanzanpassung augenscheinlich zu folgender Inkonsistenz: Die Aktivität A2 ist im Schema der Instanz I existent, sie wird aber in der Datenstruktur, welche die Zustände

speichert, nicht mehr geführt. In diesem Fall könnte der Zugriff auf die Zustandsinformationen dieser Aktivität zu einem undefinierten Verhalten des Systems führen.

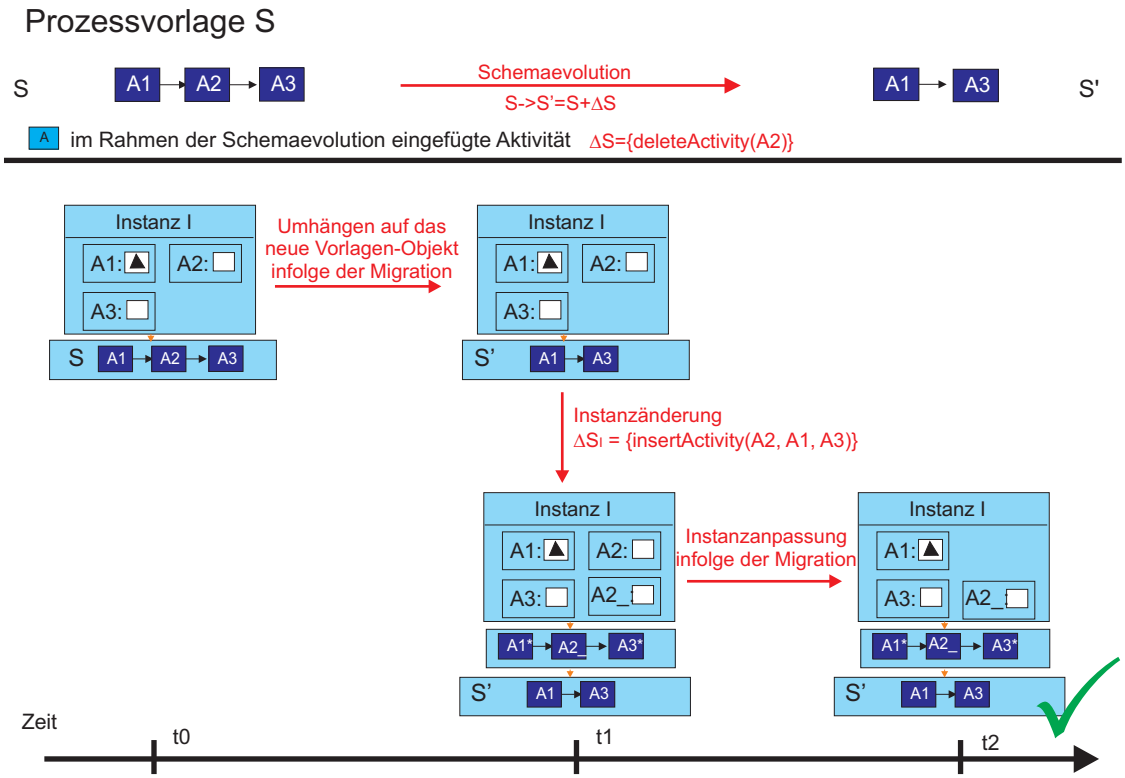

Abbildung 5.14: Konfliktlose parallel erfolgende Instanzanpassung und -änderung

In der Praxis wird dieses Problem allerdings nicht auftreten, wie Abbildung 5.14 zeigt. Auch wenn die gelöschte und die eingefügte Aktivität als dieselben betrachtet werden, so unterscheiden sie sich im System dennoch durch unterschiedliche IDs. In Abbildung 5.14 wird dies durch den zusätzlichen Unterstrich bei der im Rahmen der Ad-hoc-Modifikation eingefügten Aktivität A2 verdeutlicht. Über die ID wird im Allgemeinen in der Instanz der Zugriff auf die Zustandsinformationen einer Aktivität erfolgen. Deshalb wird mit dem erneuten Einfügen der Aktivität A2 parallel zu dem bereits existierenden ein weiterer Eintrag in der Datenstruktur angelegt, auf den mit der neuen ID der Aktivität zugegriffen werden kann. Im Rahmen der Instanzanpassung wird dann der alte Eintrag gelöscht, der neue bleibt dagegen unangetastet: Es tritt kein Konflikt auf.

Auch in der Phase der Markierungsanpassung kommt es zu keinen Problemen durch parallel erfolgende Instanzänderungen.

5.9 Zusammenfassung

Dieses Kapitel zeigt, dass bei der Schemaevolution, Migration und bei Instanzänderungen Konflikte aufgrund von konkurrierenden Aktionen auftreten können. Damit ein korrekter Ablauf dieser adaptiven Vorgänge gewährleistet ist, müssen konfliktverursachende konkurrierende Aktionen synchronisiert werden.

Viele dieser Konflikte lassen sich z. B. durch den Einsatz von Sperren vermeiden, ohne den parallelen Ablauf der konfliktverursachenden Aktionen völlig zu unterbinden. Es hat sich aber gezeigt, dass es aus pragmatischen Gründen auch sinnvoll sein kann, die Nebenläufigkeit dieser Aktionen ganz zu verhindern. Beide Strategien weisen Vor- und Nachteile auf. Welche Strategie gewählt wird, muss von Fall zu Fall abhängig vom Einsatzszenario abgewogen werden.

Damit bei dem Einsatz von Sperren die Sperrverwaltung nicht zum Flaschenhals wird, wenn viele Zugriffe darauf gleichzeitig erfolgen, muss eine von der Performance her effiziente Implementierung der Sperren gefunden werden. Die im Umfeld von Datenbanken gewonnenen Erkenntnisse über die Sperrverwaltung können hierfür hilfreich sein.

Kapitel 6

Zusammenfassung

Ziel dieser Diplomarbeit ist es, Konzepte für die effiziente Realisierung der Prozess-Schemaevolution mit anschließender Instanz-Migration in einem Hochleistungs-Prozess-Management-System vorzustellen.

Voraussetzung für eine effiziente Realisierung der Prozess-Schemaevolution mit anschließender Instanz-Migration ist, dass eine geeignete interne Repräsentationsform für Prozessvorlagen und -instanzen vorliegt, die sowohl die Schemaevolution mit der anschließenden Migration der auf der geänderten Vorlage basierenden Instanzen, als auch das dynamische Ändern einer Instanz gegenüber ihrer Vorlage auf effiziente Art und Weise ermöglicht, gleichzeitig aber auch einen geringen Speicherbedarf aufweist.

Mit der in Kapitel 3 vorgestellten Architektur wurde eine interne Repräsentationsform für Prozessvorlagen und -instanzen gefunden, die dieses Ziel erfüllt und für den Einsatz in einem Hochleistungs-Prozess-Management-System geeignet ist: Die Architektur ist in der Lage, sowohl unverzerrte als auch verzerrte Instanzen speichersparend darzustellen und garantiert eine effiziente Schemaevolution mit anschließender Migration der auf der geänderten Vorlage basierenden Instanzen.

Der entscheidende Grundgedanke bei deren Entwicklung bestand darin, alle Instanzen desselben Prozesstyps dasselbe Vorlagen-Objekt referenzieren zu lassen und das zu diesem Prozesstyp gehörige Ablaufschema alleine in diesem Vorlagen-Objekt zu halten, anstatt in jedem Instanz-Objekt repliziert. Damit kann nicht nur der Speicherbedarf gering gehalten werden, es ermöglicht auch eine effiziente Migration der verträglichen, unverzerrten Instanzen nach einer Schemaevolution. Die Instanzen müssen dazu nur von dem Vorlagen-Objekt, das die alte Version der Vorlage repräsentiert, auf das durch die Schemaevolution neu hervorgegangene Vorlagen-Objekt umgehängt werden. Mit dem Umhängen werden – logisch gesehen – die Schemaänderungen auf den Laufzeitschemata der Instanzen nachgezogen. Nicht physisch irgendwelche Schemata für eine Anpassung der zu migrierenden Instanzen auf die neue Vorlagenversion ändern zu müssen, wie es bei der Variante der Fall ist, bei der bei jeder Instanz das zu ihr gehörige Schema explizit

gespeichert wird, bewirkt einen enorm positiven Effekt auf die Effizienz von Migrationen, denn das physische Ändern eines Schemas ist gegenüber einer einfachen Referenz-Anpassung eine vergleichsweise aufwändige und teure Operation.

Für die Repräsentation einer ad hoc geänderten Instanz wurde die Delta-Schicht eingeführt. Das Delta-Schicht-Objekt repräsentiert die durch die Ad-hoc-Modifikation entstandene Abweichung der Instanz von ihrer Vorlage. Das Delta-Schicht-Objekt wird zwischen das zu der Instanz gehörige Instanz-Objekt und das Vorlagen-Objekt gehängt oder auf ein weiteres Delta-Schicht-Objekt gestapelt. Das typ-bestimmende Vorlagen-Objekt und die Delta-Schicht-Objekte bilden zusammen das (geänderte) Laufzeitschema der Instanz. Eine Alternative wurde diskutiert: Das gegenüber der Vorlage abgeänderte Laufzeitschema kann auch durch eine Kopie des entsprechenden Vorlagen-Objekts repräsentiert werden, auf dem die Instanzänderungen eingespielt worden sind. Der Nachteil dieses Ansatzes ist, dass durch jedes Kopieren nicht nur die anzupassenden Prozessabschnitte vervielfältigt werden, sondern auch die ungeänderten. Die Folge ist, dass damit die ungeänderten Prozessabschnitte u. U. mehrfach redundant im System hinterlegt werden. Der Delta-Schicht-Ansatz vervielfältigt dagegen nur die Prozessabschnitte, die dann angepasst werden und vermeidet so Redundanzen. Gegenüber der anderen Variante kann dadurch erheblich Speicher eingespart werden.

Die Migration von verzerrten Instanzen erfolgt genauso effizient wie bei unverzerrten: Die Instanz wird auf das neue Vorlagen-Objekt umgehängt. Davon abweichend muss nur noch bei Instanzen, die auch noch nach der Migration gegenüber ihrer Vorlage verzerrt sind, zwischen das Instanz-Objekt und das neue Vorlagen-Objekt ein Delta-Schicht-Objekt eingefügt werden, das den neuen Bias repräsentiert.

Basierend auf dieser Architektur wurden zwei Optimierungsansätze diskutiert, die das Ziel hatten, die Migration zu beschleunigen.

Der Grundgedanke hinter den Optimierungsansätzen ist, bei einigen Instanzen Schritte des Migrationsprozesses einzusparen.

Beim ersten Optimierungsansatz werden unverzerrte Instanzen nach ihrem Laufzeitzustand gruppiert, d. h. alle Instanzen, die denselben Zustand aufwiesen, gehören einer Gruppe an. Im Rahmen der Migration muss der Verträglichkeitstest dann nur für eine Instanz dieser Gruppe durchgeführt werden, das Ergebnis kann dann auf alle Instanzen derselben Gruppe übertragen werden. Es konnte gezeigt werden, dass sich bei der vorgestellten Implementierungsvariante sogar die Zahl der bei der Migration anfallenden Umhäng-Vorgänge von Instanzen auf das neue Vorlagen-Objekt reduzieren lässt: Referenzieren alle Instanzen einer Gruppe das zugrunde liegende Vorlagen-Objekt statt direkt nur indirekt über das gemeinsame Zustandsobjekt, dann muss bei der Migration nur das Zustandsobjekt auf das neue Vorlagen-Objekt umgehängt werden und alle das Zustandsobjekt referenzierenden Instanzen sind implizit mit umgehängt und damit auf die neue Version der Vorlage migriert. Dass der Zustand nur einmal pro Gruppe in dem Zustandsobjekt gemerkt wird und nicht explizit in allen Instanzen, hat nebenbei noch den Vorteil, dass auch die im Rahmen der Migration im Allgemeinen anfallende Instanz- und Markierungsanpassung nur einmal pro Gruppe und nicht für jede Instanz einzeln vorgenom-

men werden muss. Allerdings hat sich gezeigt, dass damit zwar die Migration der Instanzen beschleunigt werden kann, dass dies aber erheblich zu Lasten der Effizienz des Weiterschaltens von Instanzen geht, da bei jedem Weiterschalten die entsprechende Instanz auf ein neues, den neuen Zustand repräsentierendes Zustandsobjekt umgehängt werden muss. Dabei kommt es zu besonders großen Verzögerungen, wenn zu dieser Zeit das Zustandsobjekt noch nicht existiert und deshalb erst noch erstellt werden muss.

Der zweite Optimierungsansatz, die Meilenstein-Ansatz, unterteilt den Prozessgraphen in mehrere Abschnitte. Zu jedem Abschnitt wird gemerkt, welche Instanzen diesen Bereich bereits betreten, aber noch nicht überschritten haben. Alle unverzerrten Instanzen, die noch nicht den ersten Abschnitt einer Änderung betreten haben, können dann automatisch als verträglich mit den Änderungen betrachtet werden, ohne explizit eines Verträglichkeitstests unterzogen zu werden[1]. Sind die Meilensteine ausschließlich an den Stellen des Prozessgraphen gesetzt, bei denen genau ein Ausführungspfad vorliegt und nicht mehrere, parallel oder alternativ angeordnete, dann können alle Instanzen, die den ersten Bereich einer Änderung bereits überschritten haben, ohne eine Verträglichkeitsprüfung als unverträglich von der Migration ausgeschlossen werden, sofern keine Sync-Kanten im Rahmen der Änderungstransaktion eingefügt worden sind. Dadurch können auch bei diesem Verfahren für einige Instanzen Schritte des Migrationsprozesses eingespart werden, womit der gesamte Migrationsvorgang nach einer Schemaevolution beschleunigt werden kann. Allerdings geschieht dies nicht in dem Maße, wie es bei entsprechender Implementierung bei dem Verfahren der Fall ist, bei dem die Instanzen nach dem Zustand gruppiert werden: Die Meilenstein-Methode bietet nicht die Möglichkeit, mehrere Instanzen gleichzeitig auf das neue Vorlagen-Objekt umzuhängen, oder gleichzeitig die Instanz- bzw. die Markierungsanpassung durchzuführen. Gegenüber der Methode mit den Zustandsgruppierungen weist dieses Verfahren jedoch einen erheblich geringeren negativen Effekt auf die Funktion des Weiterschaltens von Instanzen auf: Instanzen müssen beim Weiterschalten nur umgehängt werden, wenn dabei ein Meilenstein überschritten wird. Trotzdem ist ein jedes Weiterschalten stets etwas verzögert, da jedesmal überprüft werden muss, ob nicht ein Meilenstein überschritten wird.

Die Verzögerungen beim Weiterschalten führten zu der berechtigten Frage, ob der Einsatz dieser Verfahren überhaupt sinnvoll ist, da das Weiterschalten von Instanzen zu den am häufigsten aufzutretenden Ereignissen bei Prozess-Management-Systemen zählt und damit besonders effizient implementiert gehört. Die Migration von Instanzen kommt verglichen dazu relativ selten vor. Als Alternative wurde vorgeschlagen, die Migration in Zeiten geringer Last des Systems zu verlegen, um damit die anderen Funktionen des Systems so wenig wie möglich zu stören.

Bei der Realisierung der Adaptivität in Hochleistungs-Prozess-Management-Systemen müssen auch die Probleme berücksichtigt werden, die durch die in einer Mehrbenutzerumgebung zwangsweise auftretenden konkurrierenden Aktionen entstehen, um eine korrekte und reibungslose Schemaevolution, Migration und Ad-hoc-Modifikation von Instanzen zu garantieren.

Kapitel 5 beschäftigte sich mit diesen Problemen, die bei dem Einsatz der in Kapitel 3 entwickelten Architektur auftreten können, und erarbeitete Konzepte, wie sie sich vermeiden oder

[1] Ausnahme: Datenelemente wurden im Rahmen der Änderungstransaktion gelöscht.

beheben lassen. Dazu wurde untersucht, inwieweit sich die drei Mechanismen der Adaptivität und zusätzlich das Weiterschalten bei der gegebenen Architektur gegenseitig und untereinander beeinflussen. Nachdem die Probleme – sofern vorhanden – identifiziert wurden, wurden Lösungsvorschläge erarbeitet, um diese von vornherein zu vermeiden oder im Nachhinein aufzulösen.

Zusammenfassend kann gesagt werden, dass mit der Architektur aus Kapitel 3 unter Berücksichtigung der in Kapitel 5 herausgearbeiteten Erkenntnisse eine für den Einsatz in Hochleistungs-Prozess-Management-Systemen geeignete Realisierungsvariante gefunden wurde, die eine effiziente und auch in einer Mehrbenutzerumgebung korrekt ablaufende Prozess-Schemaevolution ermöglicht.

Literaturverzeichnis

[Aa01] Van der Aalst, W. M. P.: Exterminating the Dynamic Change Bug : A Concrete Approach to support Workflow Change. Information Systems Frontiers 3(3), 2001, Seite 297-317

[ADK99] van der Aalst, W.; Desel, J.; Kaschek, R. (Eds.): Proc. Software Architecture for Business Process Management (SABPM '99), Workshop at the CAiSE '99, Heidelberg, Bericht 390, Juni 1999

[AJ00] Van der Aalst, W.M.P.; Jablonski, S.: Dealing with Workflow Change: Identification of Issues and Solutions. Int'l Journal of Computer Systems, Science, and Engineering, Vol. 15, No. 5, 2000, Seite 267-276

[AM00] Agostini, A.; de Michelis, G.: Improving Flexibility of Workflow Management Systems. Proc. of the Int'l Conf. on Business Process Managemant (BPM '00), LNCS 1806, Springer, 2000, Seite 218-234

[BGL03] Bausch, T; Gromer, S.; Lauer, M.: Dokumentation zum Praktikum Implementierungsaspekte von Workflow-Management-Systemen. Praktikumsbericht, Abt. DBIS, Universität Ulm, 2003

[BRD01] Bauer, T.; Reichert, M.; Dadam, P.: Adaptives und verteiltes Workflow-Management. In: [HLP01], Seite 47-66

[CCPP98] Casati, F.; Ceri, S.; Pernici, B.; Pozzi, G.: Workflow Evolution. Data and Knowledge Engineering, Vol. 24, No. 3, Januar 1998, Seite 211-238

[DR04] Dadam, P.; Reichert, M. (Hrsg.) Informatik 2004 - Informatik verbindet - Band 2: Beiträge der 34. Jahrestagung der Gesellschaft für Informatik e. V. (GI). Proceedings 34. Jahrestagung der GI, 20.-24. September 2004, Ulm, Gesellschaft für Informatik, 2004

[HLP01] Heuer, A.; Leymann, F.; Priebe, D. (Hrsg.): Proc. Datenbanksysteme in Büro, Technik und Wissenschaft, Oldenburg, März 2001. Springer Verlag, 2001

[KG99] Kradolfer, M.; Geppert, A.: Dynamic Workflow Schema Evolution Based on Workflow Type Versioning and Workflow Migration. Proc. Int'l Conf. on Cooperative Information Systems, CoopIS '99, Edinburgh, Schottland, September 1999, Seite 104-114

[LRR04] Lauer, M.; Rinderle, S.; Reichert, M.: Repräsentation von Schema- und Instanzobjekten in adaptiven Prozess-Management-Systemen. In: [DR04], Seite 555-560

[LR00] Leymann, F.; Roller, D.: Production Workflow: Concepts and Techniques. Prentice Hall PTR, 2000

[RD98] Reichert, M.; Dadam, P.: ADEPTflex - Supporting Dynamic Changes of Workflows Without Losing Control. Journal of Intelligent Information Systems, Kluwer Academic Publ., Vol. 10, No. 2, March/April 1998, Seite 93-129

[RD02] Reichert, M.; Dadam, P.: Workflow-Management-Systeme: Grundlagen, Einsatz und Implementierung. Skript zur Vorlesung, Universität Ulm, Abt. DBIS, WS 2002/2003

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

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

[RRD02] Rinderle, S.; Reichert, M.; Dadam, P.: Effiziente Verträglichkeitsprüfung und automatische Migration von Workflow-Instanzen bei der Evolution von Workflow-Schemata. Informatik - Forschung und Entwicklung, Band 17, 2002, Seite 177-197 (auch: Ulmer Forschungsberichte, Nr. 2002-01, April 2002)

[RRD03a] Rinderle, S.; Reichert, M.; Dadam, P.: On Dealing With Semantically Conflicting Business Process Changes. Ulmer Informatik-Berichte, Nr. 2003-04, Juni 2003

[RRD03b] Reichert, M.; Rinderle, S.; Dadam, P.: On the Common Support of Workflow Type and Instance Changes Under Correctness Constraints. Proc. Int'l Conf. on Cooperative Information Systems, CoopIS '03, Catania, Sizilien, Italien, November 2003

[RRD04] Rinderle, S.; Reichert, M.; Dadam, P.: Correctness Criteria for Dynamic Changes in Workflow Systems - A Survey. Data and Knowledge Engineering, 2004

[SO99] Sadiq, S.W.; Orlowska, M.E.: Architectural Considerations for Systems Supporting Dynamic Workflow Modification. In: [ADK99], Seite 118-134

[We97] Weilbach, P.: Implementierungsaspekte zur Verwaltung und Synchronisation dynamischer Änderungen in prozeßorientierten Workflow-Management-Systemen. Diplomarbeit, Universität Ulm, Abt. DBIS, 1997

[We01] Weske, M.: Formal Foundation and Conceptual Design of Dynamic Adaptations in a Workflow Management System. Proc. 34th Hawaii Int'l Conf. on System Sciences, 2001

Abbildungsverzeichnis

Anhang A

Berechnung der Anzahl möglicher Instanzzustände

In diesem Anhang soll die in Abbildung 4.3 aufgestellte Formel

$$\#(S, 1, x, y, 1) = 3 + (2x + 1)(2y + 1) - 1 + 3 - 1 \tag{A.1}$$

zur Berechnung der Anzahl $\#(S, 1, x, y, 1)$ möglicher Zustände des ebenfalls abgebildeten Prozesses S hergeleitet werden.

Für diese Betrachtung sollen die Aktivitäten jeweils nur die Zustände "INAKTIV", "AKTIVIERT", "IN AUSFÜHRUNG" und "BEENDET" annehmen können, d. h. Zustände, wie z. B. "AUSGESETZT", sollen hier nicht berücksichtigt werden. Weiter wird angenommen, dass sich die erste abzuarbeitende Aktivität bei Prozessstart sofort in dem Zustand "AKTIVIERT" befindet und nicht erst in dem Zustand "INAKTIV".

Ausgangspunkt für die Herleitung der angegebenen Formel ist die Formel zur Berechnung der Anzahl möglicher Zustände bei einem sequentiellen Graphen.

A.1 Berechnung bei Prozessen mit nur sequentiell angeordneten Aktivitäten

Bei einem Prozess S_{sequ} mit n sequentiell angeordneten Aktivitäten beläuft sich die Anzahl $\#(S_{sequ}, n)$ an möglichen Zuständen auf:

$$\#(S_{sequ}, n) = 2n + 1 \tag{A.2}$$

Dies soll mithilfe der Induktion bewiesen werden. Zuerst wird die Formel für einen Prozess S_{sequ} mit genau einer Aktivität, d. h. $n = 1$, überprüft (Induktionsanfang): Die eine Aktivität kann

nacheinander die Zustände “AKTIVIERT”, “IN AUSFÜHRUNG” und “BEENDET” annehmen. Somit können Instanzen existieren, die sich maximal in drei unterschiedlichen Zuständen befinden können. Die Anzahl an möglichen Instanzzuständen bei einem Prozess mit nur einer Aktivität beläuft sich damit auf 3. Die Formel liefert für eine Aktivität ebenfalls 3. Damit ist die Indunktionsvoraussetzung erfüllt und die Formel kann für genau n Aktivitäten als korrekt angenommen werden. Bevor durch den Induktionsschritt gezeigt werden kann, dass die Formel auch für $n+1$ Aktivitäten korrekt ist und damit generell gilt, muss das Ergebnis noch auf eine andere Weise interpretiert werden: Da bei einem Prozess mit n nur sequentiell angeordneten Aktivitäten alle Aktivitäten in gegebener Reihenfolge abgearbeitet werden müssen, bedeutet das Ergebnis auch, dass der Prozess $2n+1$ verschiedene Zustände eingenommen haben muss, bis alle n Aktivitäten abgearbeitet wurden und der Prozess als beendet gilt.

Als Induktionsschritt wird nun im folgenden statt von einem Prozess mit n sequentiell hintereinander geschalteten Aktivitäten von einem mit $n+1$ ausgegangen. Die $(n+1)$. Aktivität kann erst aktiviert werden, wenn alle n Vorgänger beendet worden sind. Dazu werden laut Induktionsvoraussetzung $2n+1$ verschiedene Prozesszustände eingenommen. Bis dann der Prozess vollständig abgearbeit ist, müssen zwei weitere Prozesszustände durchlaufen werden, einmal der, bei dem sich die $(n+1)$. Aktivität noch in Ausführung befindet und einmal der, bei dem alle $n+1$ Aktivitäten abgearbeitet wurden. Damit beläuft sich der Zahl an möglichen Instanzzuständen $\#(S_{sequ}, n+1)$ auf: $\#(S_{sequ}, n+1) = \#(S_{sequ}, n) + 2 = (2n+1) + 2 = 2(n+1) + 1$. Somit stimmt die Formel auch für Prozesse mit $n+1$ sequentiell in Reihe geschalteten Aktivitäten und ist damit generell für alle Prozesse mit nur sequentiell abzuarbeitenden Arbeitsschritten gültig, was zu beweisen war.

A.2 Berechnung bei Prozessen mit zwei parallelen Ausführungssträngen

Der Prozessgraph eines Prozesses S_{par} mit zwei parallelen Ausführungssträngen kann, wie exemplarisch in Abbildung 4.3 gezeigt, in vier Bereiche eingeteilt werden: in den Vorbereich (Bereich I), den Bereich des ersten Pfades (Bereich II), den Bereich des parallel dazu angeordneten zweiten Pfades (Bereich III) und in den Nachbereich (Bereich IV). Da Bereich II und Bereich III parallel zueinander angeordnet sind, muss bei der Berechnung der Anzahl $\#(S_{par})$ an möglichen Zuständen von S_{par} darauf geachtet werden, dass jeder mögliche Zustand des Bereichs II mit jedem möglichen Zustand des Bereichs III kombinierbar ist. Somit berechnet sich die Anzahl $\#(II + III)$ an möglichen Zuständen für den Bereich II und III zusammen genommen zu $\#(II+III) = \#(II)\cdot\#(III)$. Für den durch Bereich I, II und III definierten Prozess gibt sich die Anzahl $\#(I+II+III)$ an möglichen Zuständen zu: $\#(I+II+III) = \#(I)+\#(II+III)-1 = \#(I) + \#(II) \cdot \#(III) - 1$. Die Verringerung um 1 ist notwendig, da sonst der Prozesszustand doppelt gezählt wird, bei dem alle Aktivitäten aus Bereich I beendet, die Aktivitäten A11 und A21 aktiviert und alle weiteren Aktivitäten aus den Bereichen II und III inaktiv sind. Die Anzahl $\#(S_{par})$ an möglichen Zuständen für den gesamten Prozess S berechnet sich dann schließlich zu:

$\#(S_{par}) = \#(I + II + III + IV) = \#(I + II + III) + \#(IV) - 1$. Die erneute Subtraktion einer Eins verhindert, dass der Zustand doppelt berücksichtigt wird, bei dem alle Aktivitäten aus den Bereichen I - III beendet sind und nur die erste Aktivität aus dem Bereich IV – der Join – aktiviert ist. Mit $\#(I + II + III)$ aufgelöst lautet schließlich die Formel zur Berechnung der Anzahl $\#(S_{par})$ an möglichen Zuständen für einen Prozess S_{par} mit zwei parallelen Ausführungssträngen:

$$\#(S_{par}) = (\#(I) + \#(II) \cdot \#(III) - 1) + \#(IV) - 1 \tag{A.3}$$

Besteht bei einem Prozess S_{par_sequ} Bereich I aus w, Bereich II aus x, Bereich III aus y und Bereich IV aus z sequentiell angeordneten Aktivitäten, so kann deren jeweilige Anzahl $\#(I)$, $\#(II)$, $\#(III)$ bzw. $\#(IV)$ an möglichen Zuständen mithilfe der Formel A.2 berechnet werden. Eingesetzt in die Formel A.3 ergibt sich dann für die Gesamtanzahl $\#(S_{par\_sequ}, w, x, y, z)$ an möglichen Zuständen des Prozesses S_{par_sequ}:

$$\#(S_{par\_sequ}, w, x, y, z) = ((2w + 1) + (2x + 1)(2y + 1) - 1) + (2z + 1) - 1 \tag{A.4}$$

A.3 Berechnung bei dem Prozess aus Abbildung 4.3

In der Abbildung 4.3 liegt ebenfalls ein Prozess vor, bei dem sämtliche Bereiche für sich betrachtet sequentielle Prozessgraphen sind. Die Besonderheit liegt darin, dass der Vorbereich und der Nachbereich jeweils nur aus einer Aktivität besteht und damit w und z in der Formel A.4 auf 1 gesetzt werden müssen. Deshalb berechnet sich die Anzahl $\#(S)$ an möglichen Zuständen für diesen Prozess S zu

$$\begin{aligned} \#(S) &= \#(S, 1, x, y, 1) \\ &= ((2 \cdot 1 + 1) + (2x + 1)(2y + 1) - 1) + (2 \cdot 1 + 1) - 1 \\ &= (3 + (2x + 1)(2y + 1) - 1) + 3 - 1 \end{aligned} \tag{A.5}$$

und entspricht damit der in der Abbildung genannten Formel A.1.
q .e. d.

Anhang B

Proof-Of-Concept-Prototype

Wie in Kapitel 3.8 angesprochen, haben wir für die Validierung der aufgestellten Konzepte im Rahmen eines Praktikums und dieser Diplomarbeit den Demonstrator als einen Prototypen eines einfachen adaptiven Prozess-Management-Systems entwickelt, mit dem die Schemaevolution, die dynamische Änderung einer Instanz und die Migration von sowohl unverzerrten als auch verzerrten Instanzen demonstriert werden kann.

Im folgenden wird anhand von Beispielen dessen Leistungsspektrum aufgezeigt.

Im Rahmen einer Schemaevolution wird die in Abbildung B.1 gezeigte Prozessvorlage S der Version $V1$ durch das Einfügen der neuen Aktivität A3 hinter A2 und durch das Einfügen einer Sync-Kante zwischen die parallel angeordneten Aktivitäten A12 und A21 in die in Abbildung B.2 gezeigte Version $V2$ überführt.

Die Screenshots aus den Abbildungen B.1 bis B.6 dokumentieren den Versuch des Systems, die auf der Vorlage S der Version $V1$ basierenden, zum Teil verzerrten Instanzen I1 - I8 auf die neue Version $V2$ der Vorlage zu migrieren.

Abbildung B.1 zeigt neben der alten Version $V1$ der Prozessvorlage S die Instanzen I1-I3 vor deren Migrationsversuch mit ihren Ausführungshistorien.

Die Laufzeitschemata dieser drei Instanzen stimmen exakt mit dem Schema der Vorlage überein. Sie sind somit gegenüber ihrer Vorlage nicht verzerrt. Wie im Kapitel 2.3.1 über die Migration von unverzerrten Instanzen ausgeführt, werden unverzerrte Instanzen im Rahmen der Migration an die neue Version der Vorlage angepasst, indem die Schemaänderungen – logisch gesehen – auf den Laufzeitschemata der Instanzen nachgezogen werden. Abbildung B.2 zeigt die Resultate.

I1 wurde korrekt in dieser Weise an die neue Vorlage $S, V2$ angepasst: Ihr neues Laufzeitschema stimmt mit der neuen Version des Schemas ihrer Vorlage überein. Zusätzlich wurde korrekterweise mit der Migration in der Phase der Markierungsanpassung die Aktivität A21 von dem Status „AKTIVIERT“ in den Status „INAKTIV“ zurückversetzt: Die neue Sync-Kante $A12 \rightarrow A21$ schreibt vor, dass vor der Ausführung von A21 erst die Aktivität A12 erfolgreich beendet worden

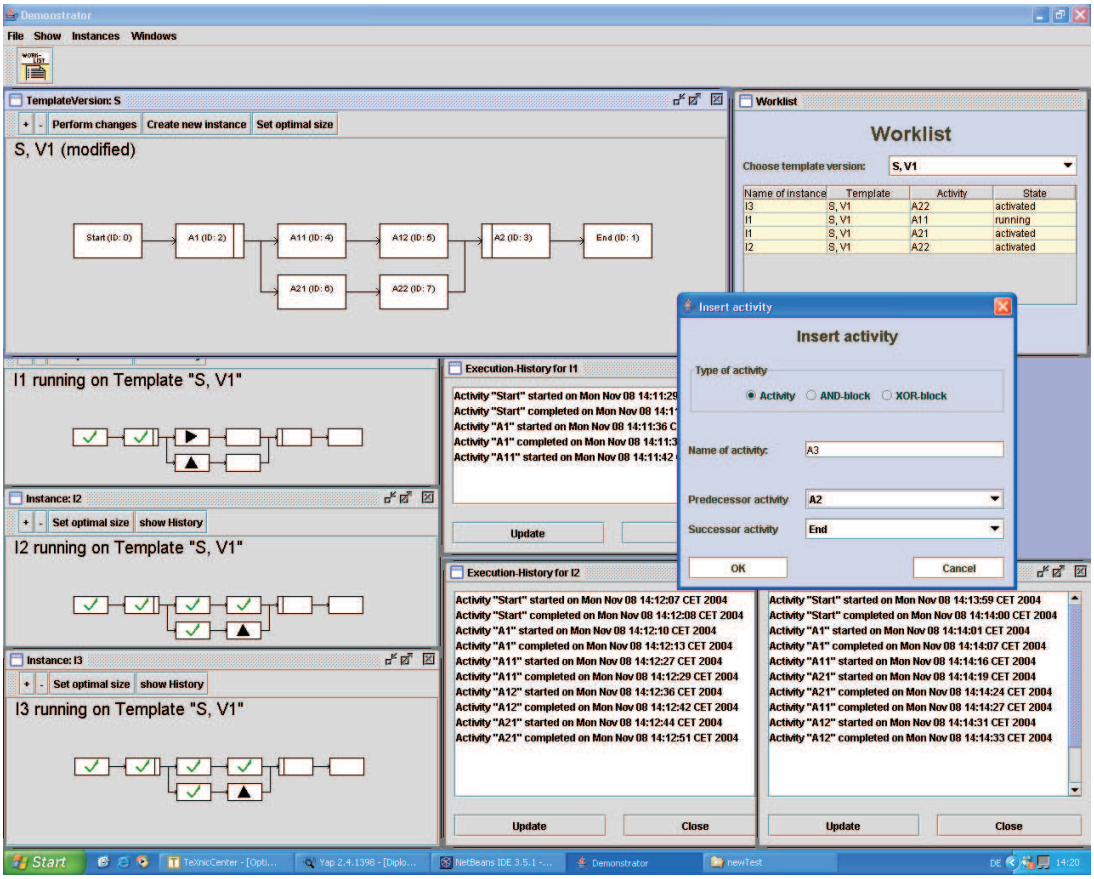

Abbildung B.1: Die Instanzen I1 - I3 vor deren Migrationsversuch

sein muss. Da aber A12 bei I1 noch nicht einmal aktiviert worden ist, darf A21 bei dieser Instanz nicht gestartet werden können. Somit war es notwendig, sie wieder zu deaktivieren und damit die entsprechenden Arbeitsaufträge aus den Arbeitslisten der infrage kommenden Bearbeiter zu entfernen.

Auch die Instanz I2 wurde korrekt auf die neue Version $V2$ ihrer Vorlage S migriert. Die Instanz-Überschrift „I2 running on Template “S, V2”“ bestätigt, dass die Instanz von nun an gemäß den Vorgaben der neuen Vorlage ablaufen wird.

Warum aber läuft die Instanz I3 immer noch nach der alten Version $V1$ der Vorlage S ab und wurde nicht auf die neue Version $V2$ migriert, obwohl sie zu Beginn denselben Zustand wie die Instanz I2 aufwies? Der in Abbildung B.2 abgebildete Migrationsreport gibt einen Hinweis: I3 ist für die durchzuführenden Modifikationen zu weit fortgeschritten. I3 ist somit unverträglich mit den Schemaänderungen. Der Grund hierfür ist bei der Einfüge-Operation der Sync-Kante zu suchen: Eine Sync-Kante kann in das Laufzeitschema einer Instanz nur eingefügt werden,

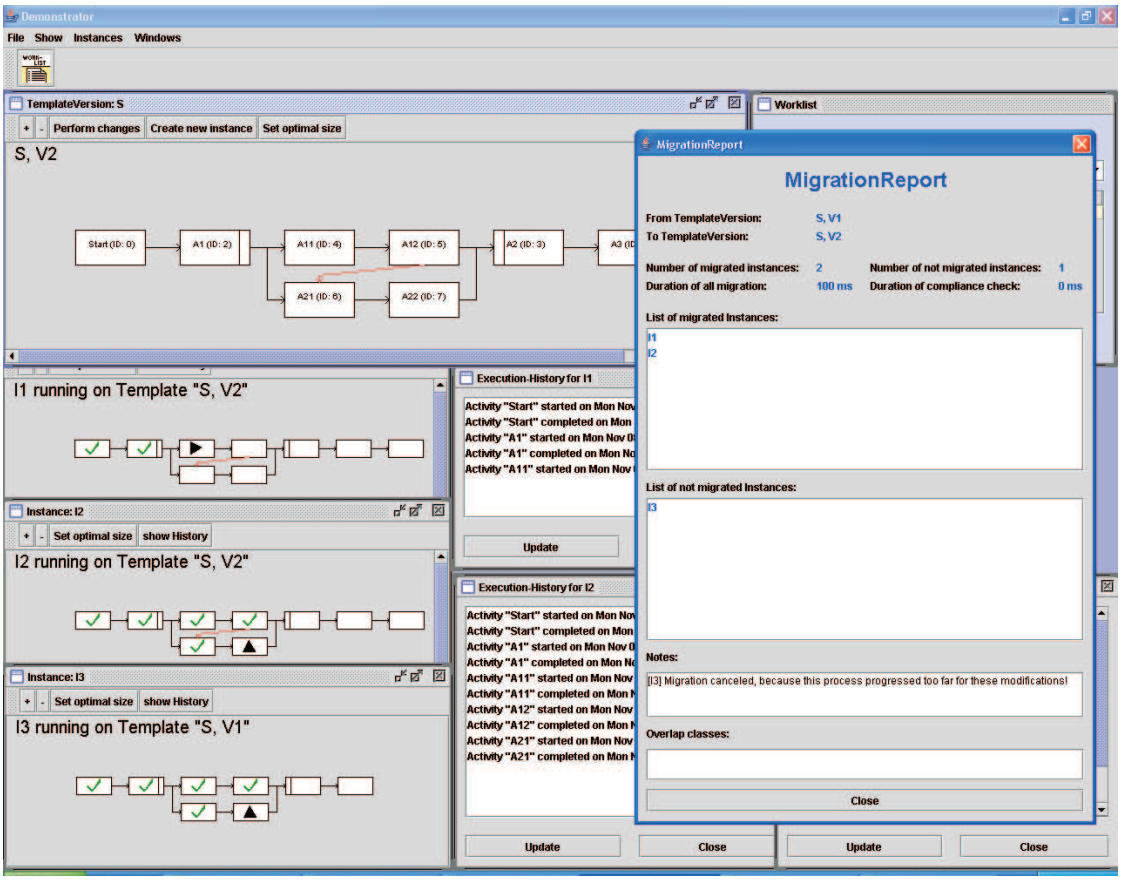

Abbildung B.2: Die Instanzen I1 - I3 nach deren Migrationsversuch

wenn die Quellaktivität entweder in einem abgewählten Zweig eines XOR-Blocks liegt oder wenn die Zielaktivität entweder höchstens aktiviert wurde oder erst nach dem Ende der Quellaktivität gestartet wurde [RRD02]. Der Demonstrator hat anhand der Ausführungshistorie von I3 richtig erkannt, dass bei dieser Instanz die Aktivität A21 vor dem Ende der Aktivität A12 zur Ausführung gelangte und dass I3 damit im Gegensatz zu I2, bei der A21 erst nach dem Ende von A12 gestartet wurde, gegen die genannte Bedingung für die Verträglichkeit mit der Sync-Kanten-Einfüge-Operation verstößt und nicht migriert werden darf.

Die im folgenden betrachteten Instanzen I4 - I8 basieren wie schon die Instanzen I1 - I3 auf der Vorlagenversion $S, V1$, sind aber alle, im Gegensatz zu diesen, gegenüber ihrer Vorlage verzerrt. Abbildung B.3 zeigt die Instanzen I4, I5 und I6 vor ihrer Migration auf die neue Schemaversion $S, V2$ zusammen mit ihren Änderungshistorien. Die Historien dokumentieren, in welcher Weise die Instanzen gegenüber ihrer Vorlage abgeändert wurden.

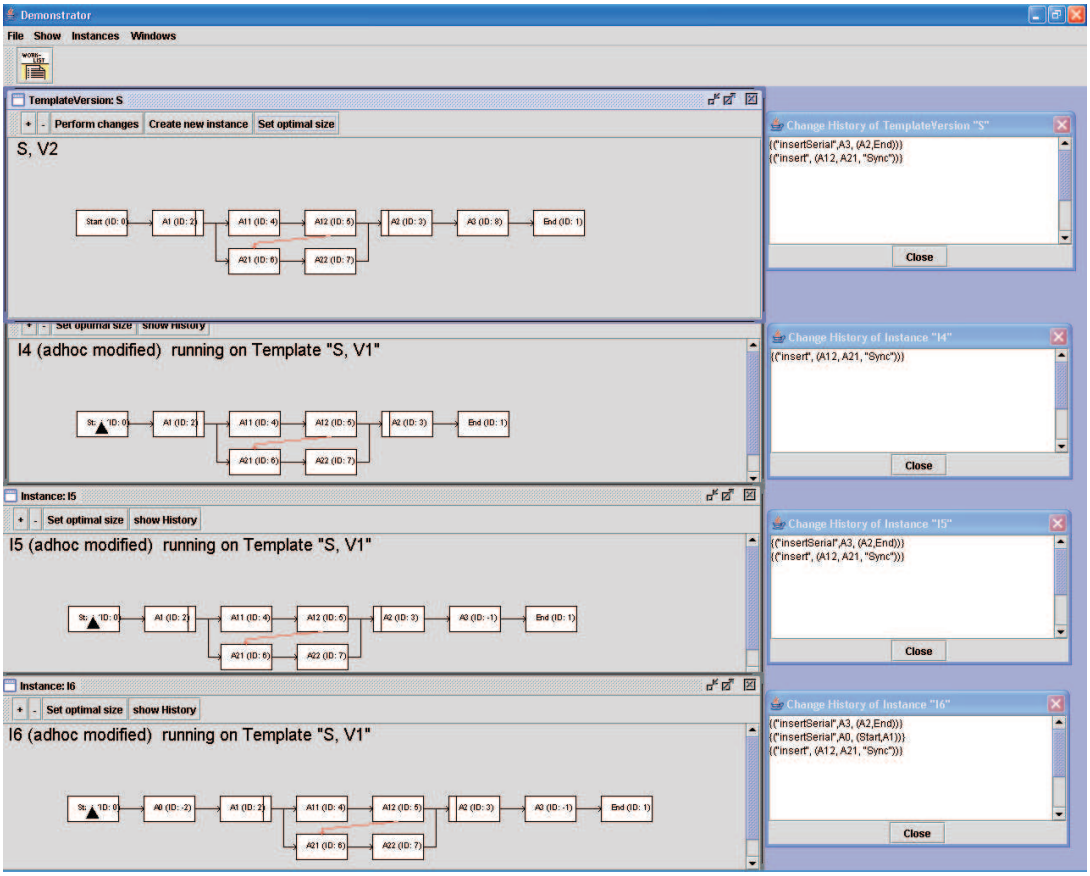

Abbildung B.3: Die Instanzen I4 - I6 vor deren Migrationsversuch

I4 weist zusätzlich die Sync-Kante $A12 \rightarrow A21$ zwischen den Aktivitäten A12 und A21 auf. I5 weicht von ihrer Vorlage $S, V1$ nicht nur durch die Sync-Kante $A12 \rightarrow A21$ ab, sondern auch durch die zusätzlich eingefügte Aktivität A3. An I6 wurden dieselben Ad-hoc-Änderungen vorgenommen wie an der Instanz I5, aber darüber hinaus wurde auch noch die Aktivität A0 vor A1 eingefügt.

Die Ergänzung „(adhoc modified)“ in den jeweiligen Instanz-Überschriften belegen, dass es sich um verzerrte Instanzen handelt.

Betrachtet man die Änderungshistorien der Instanzen und vergleicht sie mit der ebenfalls abgebildeten Historie der Vorlagen-Änderung $S, V1 \rightarrow S, V2$, dann erkennt man, dass sich bei diesen Instanzen die beiden Änderungsarten überlappen.

Sowohl bei der Ad-hoc-Modifikation der Instanz I4 als auch bei der Schemaevolution wurde eine Sync-Kante zwischen die Aktivitäten A12 und A21 eingefügt. Da an I4 keine weiteren Modifikationen vorgenommen, an der Vorlage aber weiter noch die Aktivität A3 eingefügt wurde,

ist die Instanzänderung von I4 subsumptions-äquivalent zu den Schemaänderungen. Um die Instanz I4 auf die neue Vorlage $S, V2$ zu migrieren, muss gemäß der Theorie zu der Migration von verzerrten Instanzen auf logischer Ebene dann nur noch die Aktivität A3 an dieselbe Stelle im Laufzeitschema der Instanz eingefügt werden, anstatt sämtliche Schemaänderungen auf der Instanz nachzuvollziehen.

I5 wurde laut ihrer Änderungshistorie in exakt derselben Weise wie die alte Schemaversion $S, V1$ abgeändert, d. h. bei dieser Instanz gilt, dass die Instanzänderungen äquivalent zu den Schemaänderungen sind. Dies kann auch daran erkannt werden, dass die Instanz in ihrem Laufzeitschema schon vor der Migration mit dem Schema der neuen Vorlagenversion $S, V2$ übereinstimmt. Auf logische Ebene muss somit die Instanz I5 im Rahmen der Migration nicht mehr an die neue Version ihrer Vorlage angepasst werden, sie ist es bereits.

Wie durch einen Vergleich der Änderungshistorien von der Instanz I6 und der Vorlage leicht zu erkennen ist, wurden auf Instanzebene dieselben Änderungen wie auf Typebene vorgenommen, aber zusätzlich noch die Aktivität A0 eingefügt. Damit sind die Instanzänderungen von I6 aufgrund der zusätzlichen Einfüge-Operation der Aktivität A0 umfassender als die Schemaänderungen, die Schemaänderungen sind subsumptions-äquivalent zu den Instanzänderungen. Ähnlich wie bei der Instanz I5 müssen keine zusätzliche Änderungen an der Instanz I6 vorgenommen werden, um sie an die neue Vorlage anzupassen. Sie ist bereits angepasst. Im Gegensatz zu I5 ist I6 auch noch nach der Migration verzerrt, da sie der neuen Vorlagenversion gegenüber die zusätzliche Aktivität A0 aufweist.

Die Aufgabe des Systems ist nun, den Überlappungsgrad der Instanzänderungen einer jeden verzerrten Instanz mit den Typänderungen korrekt zu bestimmen und dann die Instanz geeignet auf die neue Vorlage anzupassen. Abbildung B.4 zeigt das Ergebnis.

Wie der abgebildete Migrationsreport zeigt, wurden die Instanzänderungen von I4 korrekt als subsumptions-äquivalent und die von I5 korrekt als äquivalent zu den Schemaänderungen erkannt. Auch bei der Instanz I6 hat das System den richtigen Überlappungsgrad bestimmt.

Ebenso konnten die auf diese Informationen angewiesenen Migrationen erfolgreich von dem System durchgeführt werden: Das Laufzeitschema der Instanz I4 weist nach deren Migration die bisher fehlende Aktivität A3 auf und stimmt damit mit der neuen Version $V2$ des Vorlagenschemas S überein. Sie läuft von nun an gemäß dem neuen Schema $S, V2$ ab, wie die Instanz-Überschrift „I4 running on Template “S, V2”“ bestätigt. Das Fehlen der Ergänzung „(adhoc modified)“ lässt erkennen, dass das System die Instanz I4 auch als nicht mehr verzerrt gegenüber der neuen Version der Vorlage betrachtet und lässt darauf schließen, dass die Instanz von jetzt an auch intern wieder wie eine jede unverzerrte Instanz ohne eine Delta-Schicht repräsentiert wird.

Die neue Überschrift zur Instanz I5 zeugt einerseits davon, dass auch sie nicht mehr auf der Version $V1$ der Vorlage S beruht, sondern jetzt, wie beabsichtigt, auf der neuen Version $V2$, und andererseits, dass auch sie analog zu I4 nicht mehr verzerrt gegenüber der neuen Vorlagen-Version ist.

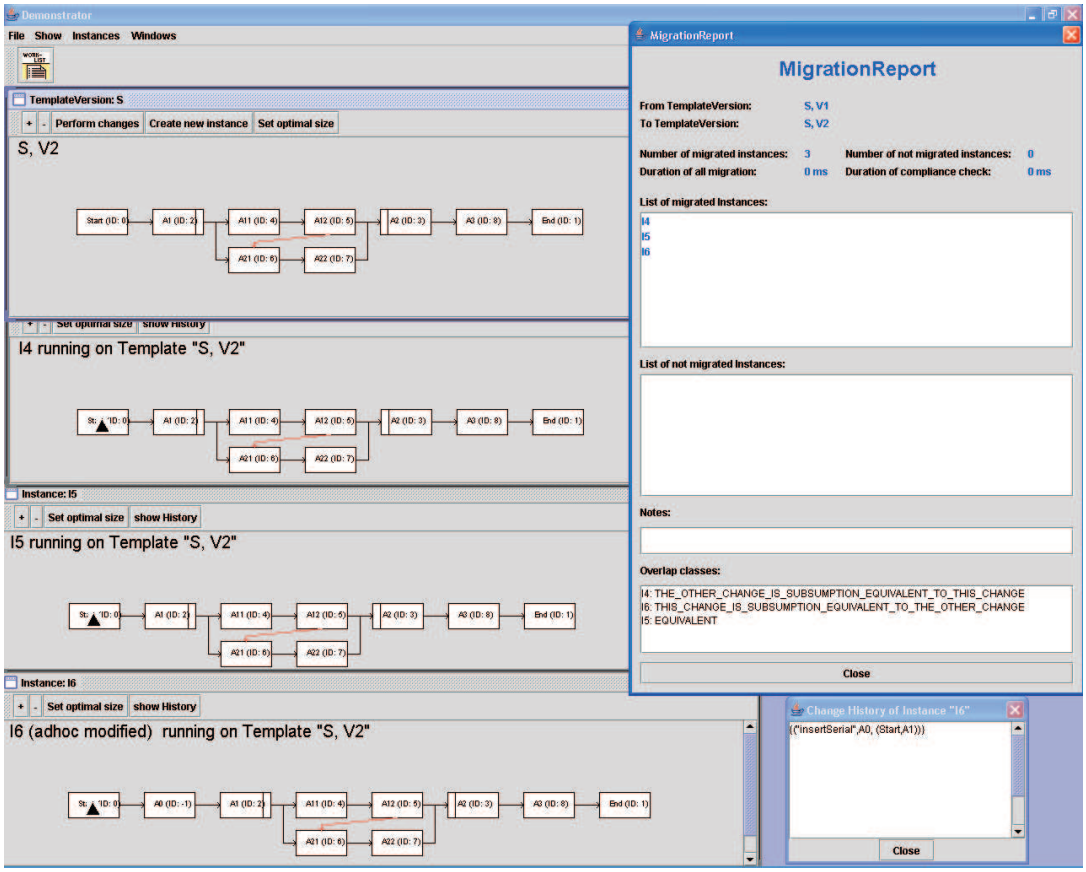

Abbildung B.4: Die Instanzen I4 - I6 nach deren Migrationsversuch

Ebenfalls an der Überschrift ist zu erkennen, dass auch Instanz I6 erwartungsgemäß auf der neuen Version $V2$ der Vorlage läuft. Das System erkannte richtig, dass diese auch nach der Migration noch verzerrt ist und hat korrekt den neuen Bias berechnet, wie die neue „Änderungshistorie“[1] der Instanz zeigt: Auf Instanzebene ist zwischen der Startaktivität des Prozesses und der Aktivität A1 die Aktivität A0 eingefügt. Im Gegensatz zu dem Zeitpunkt vor der Migration enthält diese „Historie“ folgerichtig nicht mehr die Einträge für die Sync-Kante und die Aktivität A3, da diese Elemente nun von der neuen Vorlage vorgeschrieben werden und somit nicht mehr zur Verzerrung der Instanz beitragen.

Der Demonstrator führt aber auch die Migration von verzerrten Instanzen korrekt durch, bei denen die Ad-hoc-Modifikationen disjunkt zu den Schemaänderungen sind.

[1] Der Begriff „Historie“ im eigentlichen Sinne ist hier nicht korrekt, da diese „Historie“ nicht wirklich die vom Benutzer durchgeführten Änderungen dokumentiert, sondern nur den Bias in Form einer Historie beschreibt.

Abbildung B.5 zeigt u. a. mit I7 ein Beispiel einer Instanz mit einer zu den Schemaänderungen disjunkten Modifikation.

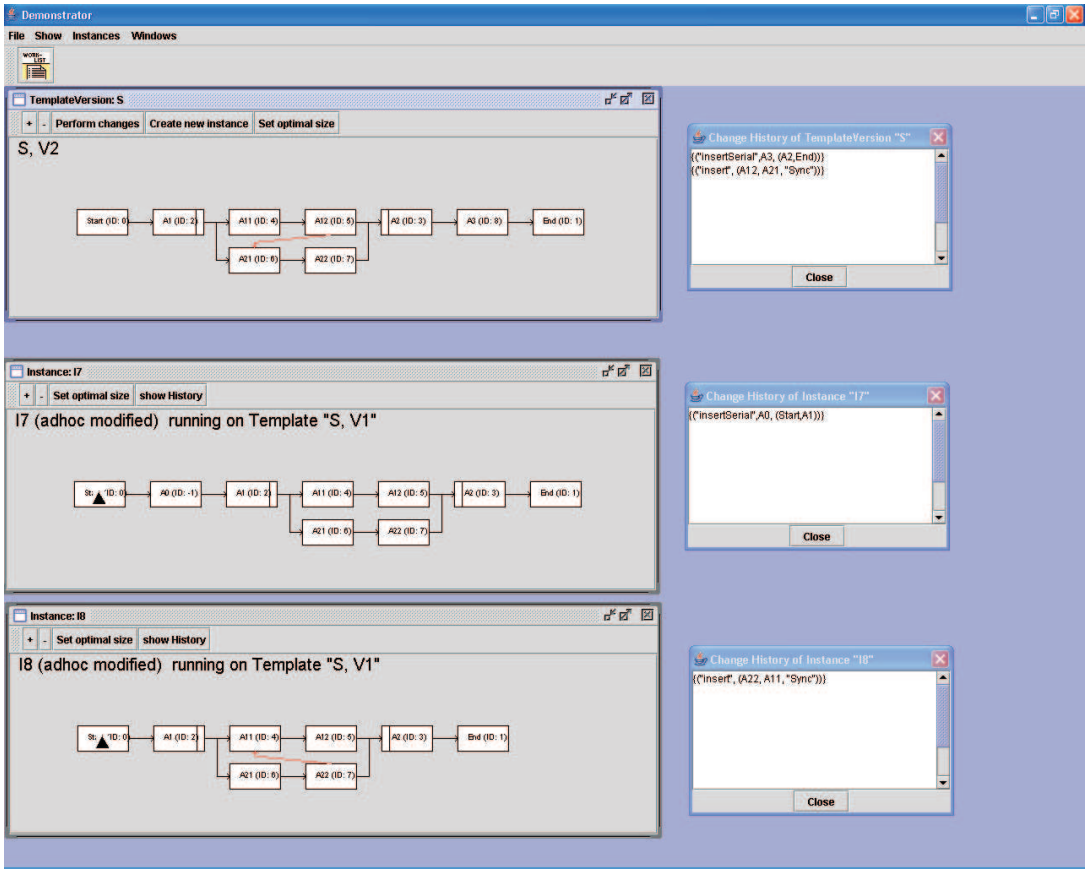

Abbildung B.5: Die Instanzen I7 und I8 vor deren Migrationsversuch

Der Theorie nach werden Instanzen mit zu den Schemaänderungen disjunkten Modifikationen an die neue Version der Vorlage angepasst, indem - auf logischer Ebene - die Schemaänderungen wie im Falle von unverzerrten Instanzen einfach auf den Laufzeitschemata der Instanzen nachgezogen werden. Das neue Laufzeitschema weist danach sowohl die auf Instanz- als auch die auf Typebene gemachten Änderungen gegenüber der alten Version des Schemas auf.

Wie Abbildung B.6 gezeigt, hat der Demonstrator dasselbe Ergebnis mit der in Kapitel 3.6 beschriebenen Methode zur Migration von Instanzen bei disjunkten Änderungsoperationen erreicht: Das Laufzeitschema der Instanz I7 weist nun sowohl die auf Schemaebene eingefügte Sync-Kante $A12 \rightarrow A21$, als auch die ebenfalls auf Schemaebene eingefügte Aktivität A3 und die auf Instanzebene eingefügte Aktivität A0 auf. Die Instanz-Überschrift teilt korrekt mit, dass I7 von nun an auf der neuen Version $V2$ der Vorlage S basiert und dass sie weiterhin verzerrt gegenüber ihrer Vorlage ist.

Vorraussetzung für die korrekte Migration war, dass der Demonstrator auch die Disjunktheit der Änderungsarten erkennt. Der Migrationsreport aus der Abbildung B.6 bestätigt dies.

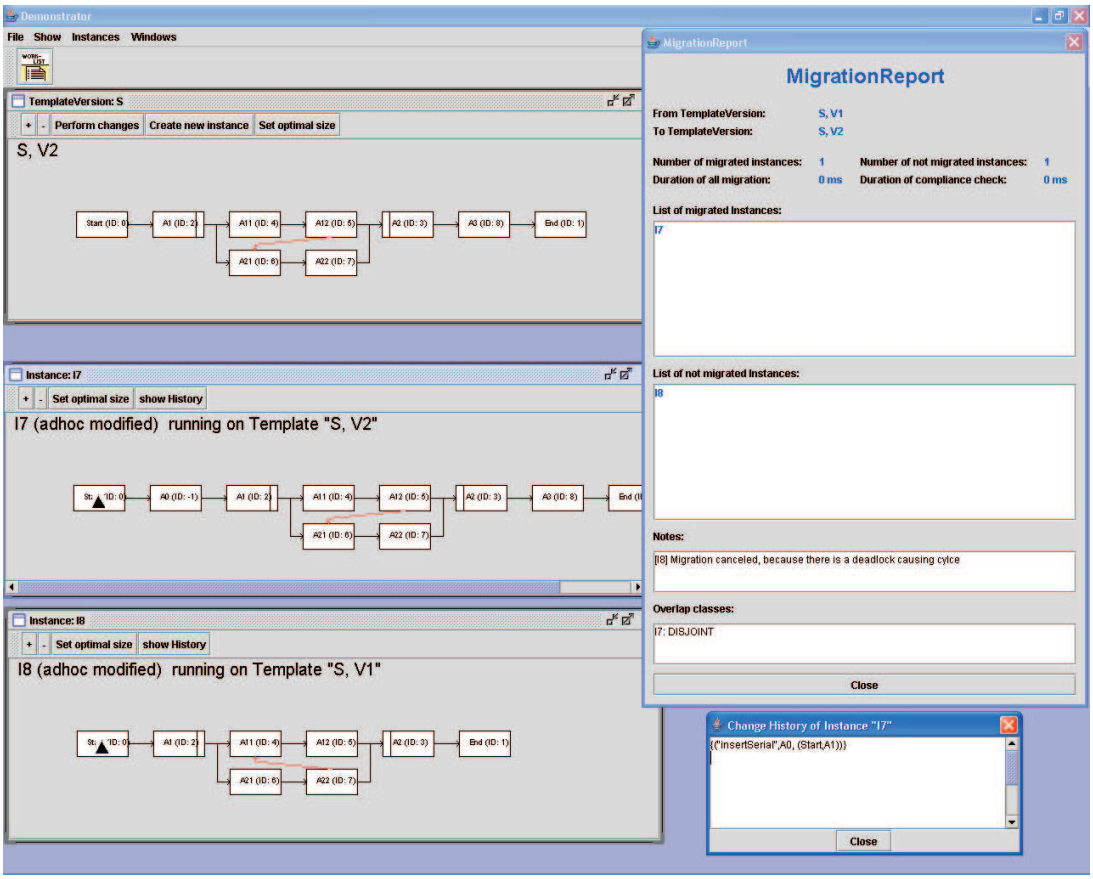

Abbildung B.6: Die Instanzen I7 und I8 nach deren Migrationsversuch

Mit dem Scheitern der Migration von Instanz I8 kann demonstriert werden, dass der Demonstrator auch in der Lage ist, die Migration von Instanzen zu verhindern, die zwar vom Zustand her verträglich mit den durchzuführenden Änderungen sind, aber bei denen die Anpassungen an die neue Vorlage zu inkorrekten Laufzeitschemata führen würden. Wie die Bemerkung in dem Migrationsreport zu dieser Instanz erahnen lässt, hätten die aus den Instanzänderungen bzw. Schemaänderungen stammenden Sync-Kanten bei I8 den Zyklus $A11 \rightarrow A12 \rightarrow A21 \rightarrow A22 \rightarrow A11$ im Kontrollfluss verursacht. Dieser Zyklus wäre dann für einen Deadlock während der weiteren Prozessausführung von I8 verantwortlich gewesen. Deshalb wurde I8 korrekterweise von der Migration ausgeschlossen.

Anhand dieser Beispiele wurde klar gezeigt, dass der Demonstrator die wichtigsten Anforderungen eines der Theorie nach voll adaptiven Prozess-Management-Systems erfüllt: Er unterstützt sowohl das Ändern von Prozessvorlagen als auch das dynamische Ändern einzelner Instanzen

zur Laufzeit. Wie in der Einführung zu dieser Diplomarbeit gefordert, versucht er nach Vorlagenänderung die auf der alten Version der Prozessvorlage basierenden Instanzen auf die neue Version anzupassen, wobei er auch die bereits durchgeführten Modifikationen der zu migrierenden Instanzen berücksichtigt. Zusätzlich sorgt er durch entsprechende Korrektheits- und Verträglichkeitstests dafür, dass die Anpassungen nicht zu Inkonsistenzen führen und dass dadurch die korrekte Prozessausführung zu jeder Zeit gewährleistet ist.

Erklärung

Markus, Lauer, Matrikel-Nr.: 427963

Ich erkläre, dass ich die Diplomarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den 17. Dezember 2004