Universität Ulm
Abt. Datenbanken und Informationssysteme
Leiter: Prof. Dr. P. Dadam

# Workflow-Management für Produktentwicklungsprozesse

DISSERTATION
zur Erlangung des Doktorgrades Dr. rer. nat.
der Fakultät für Informatik
der Universität Ulm

vorgelegt von

THOMAS BEUTER
aus Krumbach

Juli 2002

Amtierender Dekan: Prof. Dr. F. W. von Henke

Gutachter: Prof. Dr. P. Dadam
Prof. Dr. F. Schweiggert

Tag der Promotion: 5. Dezember 2002

# Vorwort

Die vorliegende Dissertation entstand zum größten Teil während meiner Tätigkeit als Wissenschaftlicher Mitarbeiter im Forschungszentrum Ulm der DaimlerChrysler AG und wurde von der Universität Ulm betreut. Durch meine Mitarbeit in verschiedenen Forschungsprojekten bei der DaimlerChrysler AG wurde ich mit umfangreichen und komplexen Praxisanforderungen aus den Entwicklungsbereichen konfrontiert, die Motivation und Ausgangspunkt für die in dieser Arbeit entwickelten innovativen Workflow-Konzepte waren. Zum Gelingen dieser Arbeit haben eine Vielzahl von Personen beigetragen, denen ich an dieser Stelle noch einmal ausdrücklich danken möchte:

Mein persönlicher Dank gilt meinem Betreuer Prof. Dr. Peter Dadam für seine Geduld und seine engagierte Unterstützung während meiner Promotionszeit. Durch sein Interesse an praxisrelevanten Fragestellungen wurde diese Arbeit erst ermöglicht. In zahlreichen konstruktiven Diskussionen hat er entscheidend zum inhaltlichen Gelingen der Arbeit beigetragen. Darüber hinaus habe ich von seiner großen wissenschaftlichen Erfahrung und seiner problemorientierten Sichtweise stark profitiert. Seine wissenschaftlichen Hinweise waren immer sehr willkommen und hilfreich.

Ebenso möchte ich Prof. Dr. Franz Schweiggert herzlich danken. Er hat sich zum einen als Gutachter zur Verfügung gestellt, zum anderen waren die fachlichen Gespräche mit ihm äußerst angenehm, motivierend und zielführend.

Beiden Professoren möchte ich auch für die Betreuung mehrerer Diplomarbeiten danken, die im Kontext dieser Arbeit entstanden sind, um verschiedene Teilaspekte detaillierter zu beleuchten und die Arbeit als Ganzes abzurunden.

Ein ganz besonderer Dank gilt auch meinen jetzigen und ehemaligen Vorgesetzten und Kollegen, insbesondere Thomas Bauer, Georg Grütter, Christian Heinlein, Alfred Katzenbach, Reiner Knoll, Erich Müller, Manfred Reichert, Peter Schneider, Reiner Siebert, Michael Weitner und Robert Winterstein. Meinen Vorgesetzten danke ich für die Gewährung des nötigen wissenschaftlichen Freiraums und für ihr Vertrauen in meine Arbeit. Sie gaben mir damit die große Gelegenheit, die gesammelten Praxisanforderungen wissenschaftlich aufzuarbeiten. Bei den Kollegen möchte ich mich besonders für ihre wissenschaflichen, zum Teil recht hitzigen Diskussionen und für ihre „Durchhalteparolen“ in schweren Zeiten bedanken.

Schließlich gilt mein Dank allen Diplomanden, Praktikanten und wissenschaftlichen Hilfskräften, die durch ihre Mitarbeit auch zum Gelingen dieser Disseration beigetragen haben. Jochen Rütschlin danke ich für seine LaTeX-Unterstützung, meinem Schwiegervater, Michael Wagenhuber, für das äußerst sorgfältige Korrekturlesen der Arbeit.

Danken möchte ich auch meinen Eltern, die mir neben vielen anderen Dingen die Möglichkeit für mein Informatik-Studium gegeben haben, ohne das diese Arbeit nicht hätte entstehen können.

Der größte Dank gebührt jedoch meiner Frau Daniela sowie meinen beiden Kindern Micheal und Annika, die beide im Laufe dieser Arbeit zu uns stießen. Ihre Liebe, Rücksichtnahme und Unterstützung haben diese Arbeit erst ermöglicht.

Ulm, im Juli 2002

# Kurzfassung

Workflow-Management-Systeme unterstützen die elektronische Abwicklung von Arbeitsprozessen, sogenannten Workflows. Heutige Systeme erzwingen meist die starre Ausführung vormodellierter Workflows, was ihre praktische Tauglichkeit auf Anwendungsgebiete mit starren Prozessen beschränkt. In vielen Anwendungsbereichen sind Arbeitsprozesse jedoch meist nur semi-strukturiert. Dieses Merkmal ist aufgrund kreativer Teilaufgaben auch typisch für alle Produktentwicklungsprozesse, für die in dieser Arbeit adäquate Modellierungs- und Ausführungskonzepte vorgestellt werden. Diese Konzepte wurden im **WEP**-Workflow-Management-Systems (**WEP** = Workflow Management for Engineering Processes) zur Demonstration der Machbarkeit und des praktischen Nutzens prototypisch implementiert.

Neben der adäquaten Steuerung semi-strukturierter Prozesse sind als weitere wichtige Anforderungen von Produktentwicklungsprozessen unter anderem die Unterstützung eines prozesskoordinierten Simultaneous-Engineerings zur Reduzierung von Prozessbearbeitungszeiten, die dynamische Anpassung der Prozessstruktur entsprechend aktueller (Teil-)Produktausprägungen sowie die Berücksichtigung übergeordneter Projektmeilensteine bei der Workflow-Koordination zu nennen.

Die Unterstützung dieser Anforderungen darf allerdings nicht die Datenkonsistenz (z. B. durch falsche oder veraltete Informationen) oder die Prozesssicherheit (z. B. durch Blockierungen oder Endlosschleifen) gefährden, da solche Fehler im Extremfall eine Gefahr für die Existenz eines gesamten Unternehmens darstellen können.

Die wichtigsten Bausteine von **WEP**-Workflows sind zielorientierte Aktivitäten. Sie ermöglichen durch ihre ergebnisorientierte Modellierung eine adäquate Beschreibung unstrukturierter Teilprozesse. Darüber hinaus erlaubt das Konzept der zielorientierten Aktivitäten die kontrollierte Weitergabe vorläufiger Daten bestimmter Qualität, wodurch das geforderte Simultaneous-Engineering entlang der Entwicklungsprozesskette realisiert wird. Es wird gezeigt, dass zielorientierte Aktivitäten mittels erweiterter Kontroll- und Datenflusskonstrukte zu einem strukturierten Gesamtprozess verknüpft werden können, wodurch die Modellierungskomplexität für den Anwender auch bei umfangreichen Prozessen beherrschbar bleibt.

Unstrukturierte Teilprozesse und Simultaneous-Engineering erfordern natürlich auch eine neue und weitaus komplexere Schaltlogik eines Workflow-Management-Systems. Ein weiterer Schwerpunkt der Arbeit bildet deshalb die Spezifikation eines formalen Workflow-Ausführungsmodells und der Nachweis seines korrekten Schaltverhaltens.

Eine vergleichende Diskussion verwandter Ansätze aus den Bereichen Prozess-, Daten- und Projektmanagement rundet die Arbeit ab.

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Teil I

# Motivation

# Kapitel 1

# Einführung in den Anwendungsbereich

In vielen Geschäftsbereichen wird ein komplexer Vorgang (Geschäftsprozess, Ablauf) nicht von einem Mitarbeiter allein, sondern in Zusammenarbeit mit Mitarbeitern der eigenen oder anderer Abteilungen und Unternehmen durchgeführt, die im Rahmen dieses Vorgangs Teilaufträge erfüllen. Beispiele hierfür sind Reiseplanung und -abrechnung, Untersuchungszyklen in Kliniken, Kreditanträge und -anpassungen, Versicherungsantragsbearbeitung oder Entwicklungsprozesse in der Fahrzeugindustrie. Die Koordination dieser zum Teil komplexen und räumlich weit verteilten Abläufe wird von klassischen Informationssystemen (datenbankbasierten Anwendungen) nur sehr unzureichend unterstützt, da sie zur *statischen* Verwaltung von Informationen entwickelt wurden. Der *dynamische* Teil dieser Abläufe, wie das Zuteilen neuer Aufträge, der Austausch von Informationen zwischen Mitarbeitern oder die Festlegung und Überwachung zeitlicher Beschränkungen, erfolgt deshalb häufig informell zwischen den einzelnen beteiligten Personen. Dadurch wird ein erheblicher Zusatzaufwand verursacht. Werden solche Absprachen vergessen oder nicht weitergeleitet, so führt dies zu Fehlern.

Will man mit konventionellen Programmiermethoden auch beim dynamischen Teil der Abläufe IT–Unterstützung bieten, so führt dies zu sehr komplexen Anwendungsprogrammen: Der gesamte Informations- und Kontrollfluss, einschließlich der in verteilten Anwendungen sehr aufwändigen Fehlerbehandlung, muss explizit ausprogrammiert werden. Hinzu kommt, dass diese „hart verdrahteten“ Abläufe nur schwer zu validieren, zu warten sowie an organisatorische oder funktionale Änderungen anzupassen sind.

Workflow-Management-Systeme [GHS95, Jab95, JB96, SB96, Dei97] bieten hier durch Trennung von Ablauflogik (*dynamisch*) und eigentlichem Anwendungscode (*statisch*) einen vielversprechenden Ansatz. Durch „Herausziehen“ der Ablauflogik aus dem Anwendungscode kann ein Ablauf schnell spezifiziert, über Analyse beziehungsweise Animation validiert und gegebenenfalls rasch an neue oder veränderte Anforderungen angepasst werden.

Ein Workflow-Management-System[1] besteht aus einer *Buildtime-Komponente* zur Modellierung und Validierung von Abläufen, sogenannten *Workflows*, und einer *R*untime-Komponente für deren Ausführung (Laufzeitumgebung). Die Runtime–Komponente eines Workflow-Management-Systems besteht wiederum aus einer oder mehreren *Workflow-Engines*, welche die eigentliche Ausführung der

[1] Im Bereich der Workflow-Management-Systeme existiert eine verwirrend hohe Zahl verschiedener Terminologien (siehe [WfMC99] für eine Gegenüberstellung der Begriffe). Die hier verwendeten Begriffe orientieren sich am Referenzmodell der *Workflow Management Coalition (WfMC)*, einem Standardisierungsgremium bestehend aus führenden Herstellern von Workflow-Management-Systemen [WfMC95].

Workflows übernehmen. Die *Architektur* eines Workflow-Management-Systems legt die Zahl und das Zusammenspiel der für eine Workflow–Ausführung benötigten Workflow-Engines, ihre funktionale Aufgliederung in Komponenten (Kontrollflusssteuerung, Organisationsverwaltung) sowie ihre physikalische Verteilung auf Rechnerknoten fest.

Aufgrund ihres generischen Ansatzes können Workflow-Management-Systeme im Prinzip zur Modellierung und Ausführung von Abläufen aus beliebigen Anwendungsgebieten eingesetzt werden [RD00]. Neben dem „klassischen“ Bereich der Büroautomatisierung [Gru93, VE92, LA94] werden in letzter Zeit Workflow-Management-Systeme auch in anderen Gebieten eingesetzt. Beispiele hierfür sind medizinisch–organisatorische Abläufe [DHL91, DKR+95, RHD98, MR99], Software–Entwicklungsprozesse [ADH+92, Obe94a, Gru93] oder Prozesse in der Produktentwicklung [NW98, Kes96, MHR96, HB97].

Die verschärfte Wettbewerbssituation auf dem weltweiten Automobilmarkt zwingt die Automobilhersteller, ihre Produktentwicklungszeiten massiv zu verkürzen bei gleichzeitiger Verbesserung der Innovationsrate und der Produktqualität. Bei internen DaimlerChrysler Prozessanalysen wurde erkannt, dass hierbei der Einsatz von Workflow-Management-Technologien einen wichtigen Beitrag leisten kann, da die am Entwicklungsprozess beteiligten Personen immer nach dem gleichen Schema zusammenarbeiten [EPW94, EWM+98]. Dabei führen zum einen die Prozessbeteiligten einen hohen, nicht wertschöpfenden Anteil an administrativen Tätigkeiten durch, der teilweise bis zu 70% ihrer Arbeitszeit [Har94, Hal01] kostet. Zum anderen verursacht die starke Arbeitsteilung vielfach einen schlechten Informationsfluss mit hohen „Liegezeiten“ [Ehr95, Het00]. Beide Problemfelder können durch Workflow-Management stark abgemildert werden, da es die erforderlichen Aufgaben automatisiert. Es wurde bei den Analysen ebenfalls festgestellt, dass die bisherige Workflow-Management-Technologie an vielen Stellen nicht ausreicht, um den spezifischen Anforderungen aus der Produktentwicklung zu genügen [BD96a, BKJ+99, HB97, RM97, NW98]. In der DaimlerChrysler Forschung wurde deshalb 1999 ein Forschungsprojekt initiiert mit dem Ziel, Konzepte für eine adäquate Unterstützung von Produktentwicklungsprozessen zu entwickeln.

# Kapitel 2

# Produktentwicklungsprozesse

Produktentwicklungsprozesse im Automobilbereich sind geprägt durch die *inkrementelle Konkretisierung* einer Idee zu einem fertigen Produkt (vgl. Abbildung 2.1). Aufgrund ihrer Komplexität wird eine Entwicklungsaufgabe dabei immer in Teilaufgaben aufgegliedert. Jede Teilaufgabe konkretisiert ein *zusammenhängendes Teil* (z. B.: Motor, Tür, Bremssystem) des Produkts. Dabei werden neben der Festlegung, welche Teile eines Produkts in einer Teilaufgabe konkretisiert werden sollen, *Meilensteine* definiert, die Vorgaben über den zeitlichen Verlauf der jeweiligen Teileentwicklung spezifizieren. Bei der Entwicklung der Mercedes-Benz LKW-Baureihe Atego werden beispielsweise pro Jahr ca. 14.000 Teilaufgaben, sogenannte *Projekte* oder *Arbeitspakete*, definiert und abgearbeitet. An der Bearbeitung einer Teilaufgabe sind immer verschiedene Personen und Abteilungen beteiligt, die ihre Zusammenarbeit koordinieren müssen.

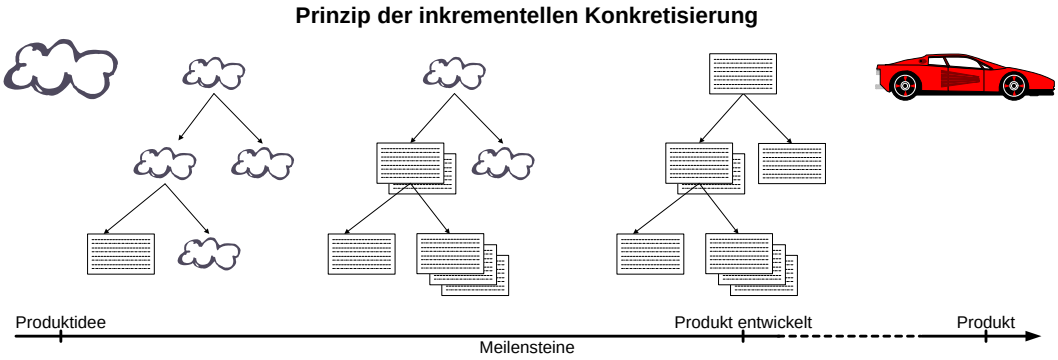



**Abbildung 2.1:** Das Prinzip der inkrementellen Konkretisierung

Ein stark vereinfachter Änderungsprozess soll als repräsentativer Vertreter aus Anwendersicht die Anforderungen von Produktentwicklungsprozessen an eine IT-Unterstützung beschreiben. Wie in Abbildung 2.2 skizziert, beginnt der Prozess mit dem Arbeitsschritt *DefineChangeRequest*, der die zu ändernden Bauteile festlegt. Für jedes dort im *Änderungsauftrag* (*changeRequest*) festgelegte Bauteil wird der verantwortliche Entwicklungsbereich informiert, in dem dann die Entwicklungsaufgabe *DesignPart* abgearbeitet wird.

Wenn alle im Änderungsauftrag aufgeführten Bauteile konstruiert worden sind, wird im Prozessschritt *CheckDMU* an einem virtuellen Prototypen (DMU = Digital Mock-Up) überprüft, ob die Bauteile in

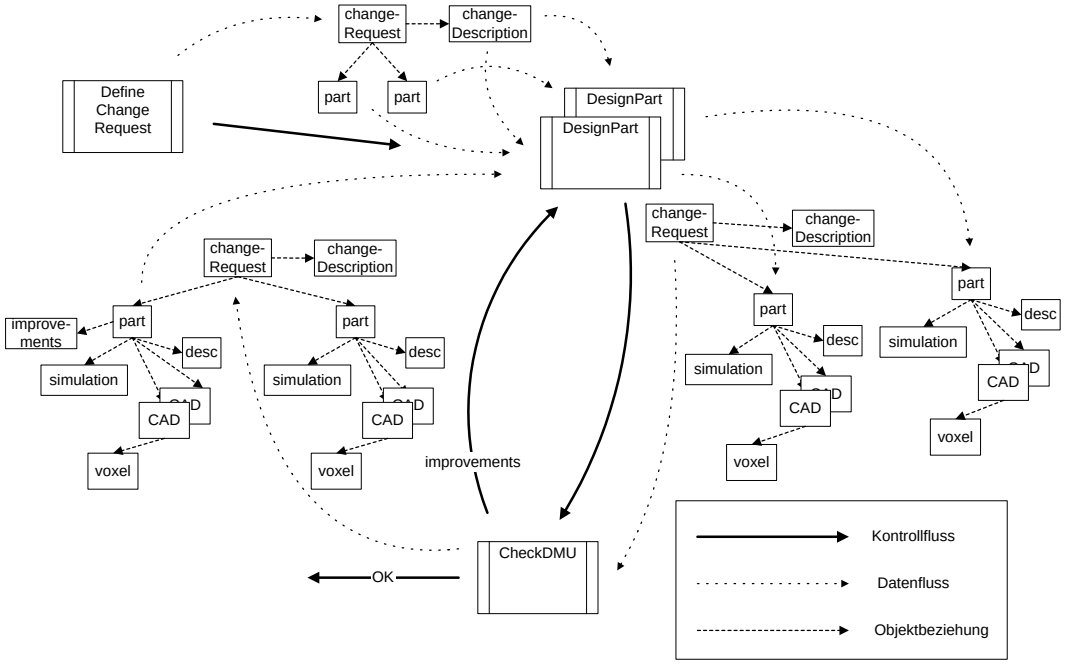



**Abbildung 2.2:** Änderungsprozess als Repräsentant von Produktentwicklungsprozessen

ihrer neuen Form „zusammenpassen“. Diese Prüfung kann viele Aspekte wie beispielsweise räumliche, funktionale, kinematische oder einbautechnische Aspekte umfassen.

Haben alle Bauteile die Prüfungen erfolgreich bestanden, so ist der Änderungsprozess beendet und nach weiteren Tests (z. B. an realen Prototypen) können die neuen Bauteilmodelle an die Produktion weitergegeben werden. Andernfalls wurden während der DMU-Prüfung Verbesserungsvorschläge (*improvements*) an die betroffenen Bauteile angehängt. Diese Bauteile müssen dann in einer weiteren der soeben skizzierten Iterationsschleife nachgebessert werden.

Leider spiegelt der bisher durch Kontroll- und Datenfluss spezifizierte Prozess nur unzureichend die Zusammenarbeit der verschiedenen involvierten Bereiche bei der Abarbeitung eines Änderungsauftrags wider: Zwar wird jeder Änderungsauftrag entlang dieser groben Prozessstruktur abgewickelt, die Interaktionen sind jedoch weitaus diffiziler und dynamischer. Um dies zu veranschaulichen, wird nun ein fiktiver Änderungsantrag *Entwicklung einer Guard-Variante* „durchgespielt“:

*Für die Entwicklung einer* Guard-Variante *müssen die Türen eines Fahrzeugs schusssicher werden. Der Verantwortliche für die Türen muss dazu im Prozessschritt DefineChangeRequest möglichst präzise die zu ändernden oder neu zu entwickelnden Bauteile spezifizieren.*

**Bereitstellung von Wissen früherer Entwicklungen**

Für eine möglichst optimale Auswahl der Bauteile ist es notwendig, auf Erfahrungen ähnlicher Entwicklungsaufgaben aus zum Beispiel anderen Fahrzeugbaureihen zurückgreifen zu können. Dieses *personenübergreifende Wissensmanagement* nimmt bei ständig komplexer werdenden Produkten einen immer höheren Stellenwert ein [KM94, Gro92, GGHV95].

Wissensmanagement wird jedoch vorhandene Systeme (z. B. CAD- und Berechnungsprogramme) keinesfalls ersetzen, sondern soll in Kombination mit diesen Systemen zur Lösung qualitativer Probleme, wie beispielsweise der Auswahl von Konstruktionsprinzipien oder von (genormten) Bauelementen beitragen [Gro92].

Wissensmanagement muss anwendungsspezifisch angeboten werden. Dabei darf die Wissensaquisation keinen oder nur einen vertretbar geringen Mehraufwand für die am Prozess beteiligten Personen verursachen. Erste vielversprechende Ansätze existieren bei DaimlerChrysler in Form von *Eboks (Engineering books of knowledge)*, in denen fahrzeugübergreifend Probleme und deren Lösungsverfahren den Entwicklungsbereichen zur Verfügung gestellt werden.

**Unterstützung global agierender interdisziplinärer Arbeitsgruppen**

Der Prozessschritt *DefineChangeRequest* wird außerdem nicht von einer Person allein durchgeführt, sondern häufig *interdisziplinär* mit anderen betroffenen Bereichen und Partnern (Konstruktion, Berechnung, Simulation, Fertigungsplanung, Werkzeugbau, Kostenbewertung, . . . ) bearbeitet, um auch ihre Belange im Änderungsprozess möglichst früh einzubinden. Die immer weiter fortschreitende Globalisierung erfordert hier neue Modelle der Zusammenarbeit, wie beispielsweise virtuelle runde Tische [TSMB95, DCSCW98] oder *dynamische Parameternetzwerke* [RCS00, GS00], um auch räumlich verteilte Personen bei der Festlegung des Änderungsumfangs integrieren zu können.

*Bei der Guard-Variante wird festgelegt, dass für eine schusssichere Tür die Scheibe verdickt werden muss. Dies hat zur Folge, dass der Türrahmen zur Aufnahme der dickeren Scheibe verbreitert und der mechanische Fensterheber verstärkt werden muss. Als Ergebnis des Prozessschritts DefineChangeRequest wird ein Änderungsauftrag changeRequest mit einer Beschreibung (changeDescription) der geforderten Änderungen und mit drei Referenzen auf die Serienbauteile Scheibe, Türrahmen und Fensterheber erzeugt (parts). Es ist somit ein erster, häufig noch vorläufiger Änderungsumfang festgelegt. Die betroffenen Entwicklungsbereiche werden parallel mittels Prozessschritt DesignPart informiert und mit aktuellen Daten versorgt. Im konkreten Fall müssen also die Bereiche für den Rohbau Tür und den Fensterheber sowie der Zulieferer für die Scheibenkonstruktion informiert werden. Der Änderungsprozess besitzt damit* momentan *drei parallele Zweige.*

**Koordination eines weltweiten, organisationsübergreifenden Entwicklungsverbunds**

Wie man am Beispiel der Scheibe sehen kann, werden auch Konstruktionsaufgaben an externe Entwicklungspartner vergeben, so dass die verschiedenen Entwicklungsbereiche in der Regel (weltweit) verteilt sind [BSH99]. Diese *organisationsübergreifenden Entwicklungsverbunde* erfordern einen klar definierten, standardisierten Prozess [Ehr95]. Eine *rein manuelle Koor-*

*dination* dieses Prozesses ist aufwändig und fehleranfällig. Ohne *Systemunterstützung* kann die Steuerung dieser dynamischen, weltweiten und organisationsübergreifenden Prozesse, insbesondere in einem E-Business-Umfeld mit wechselnden Entwicklungspartnern, nicht oder nur mit sehr hohem administrativen Aufwand bewältigt werden [Sof01].

**Unterstützung unstrukturierter Teilprozesse innerhalb vorgegebener Prozessstrukturen**

Das Ergebnis eines *mehrere Wochen bis Monate dauernden* Konstruktionsprozesses ist unter anderem eine neue Bauteilversion mit Beziehungen zu neuen CAD-Modellen in unterschiedlichen Ausprägungen, Simulationsprotokolle für den Nachweis geforderter Eigenschaften sowie einer textuellen Beschreibung des abgeänderten Bauteils. Um die benötigten Datenobjekte zu erzeugen, bedient sich ein Konstrukteur verschiedener Entwicklungswerkzeuge, beispielsweise CAD-Systeme, Text-Editoren, Simulationswerkzeuge. Diese eigentliche Konstruktionsaufgabe stellt eine sehr *kreative* Arbeit dar [Pah99]. Deshalb ist die Reihenfolge und Häufigkeit der einzelnen Entwicklungsschritte innerhalb des Subprozesses *DesignPart* ebenso wie die Zahl der erzeugten Objektversionen nicht vorherbestimmbar. Sie hängt auch stark vom einzelnen Konstrukteur, seiner konstruktionsmethodischen Ausbildung und nicht zuletzt von seinem individuellen Erfahrungsschatz ab [Pah99, Ehr95]. Es kann allenfalls ein generelles phasenorientiertes Vorgehen vorgegeben werden (vergleiche VDI-Richtlinie 2221 mit den sich überlappenden Phasen *Klären und Präzisieren der Aufgabenstellung*, *Konzipieren*, *Entwerfen*, *Ausarbeiten* [VDI86]), an dem sich ein Konstrukteur orientieren kann [Fri88]. Üblicherweise wird ein Konstrukteur mehrere Zwischenversionen als Varianten erzeugen, bis er aus der Menge der entstandenen Versionen eine Objektversion als die Endgültige markiert und mit der Weitergabe dieser Version seinen Konstruktionsprozess beendet.

Notwendig ist deshalb eine *flexible* Unterstützung *unstrukturierter, kreativer* Prozessschritte [Pah99, Fri88], die in einen vorgegebenen übergeordneten und standardisierten Prozess eingebettet sind [Fra97, Ehr95]. Dieser übergeordnete Prozess muss externen und internen Prozessstandards, wie beispielsweise ISO 9000 Zertifizierung [DIN00], Unternehmensvorgaben oder den Forderungen der Qualitätssicherung [VDI96], genügen und diese garantieren. Eine ideale Prozesssteuerung koordiniert dabei ohne den Anwender kreativitätshemmend in ein vorgegebenes Prozessschema zu pressen. Nur so lässt sich der Zielkonflikt zwischen Kreativität des Einzelnen und einer effizienten, transparenten und zielgerichteten Koordination aller Prozessbeteiligten gewährleisten.

**Prozesskoordiniertes Simultaneous-Engineering**

Bei langdauernden Prozessschritten werden in der Regel verschiedene Ergebnisdatenobjekte erzeugt. So werden zum Beispiel beim Prozessschritt *DesignPart* unter anderem 3D-CAD-Modelle und Textdokumente bereitgestellt. Ein Teil dieser Daten fällt bereits vor Ende des Prozesses an [Gro92, NW98]. Zur *Verkürzung des Gesamtentwicklungsprozesses* ist deshalb ein Bearbeiter angehalten, bereits *vorhandene endgültige Teildaten* sowie *sinnvolle Vorabversionen* anderer Ergebnisdaten vor Beendigung des Konstruktionsprozesses an parallele oder nachfolgende Schritte weiterzu-

geben, da diese häufig mit Teildaten oder auch mit vorläufigen Daten ihre Arbeit beginnen können [Sof01, Het00].

Bei dem skizzierten Änderungsprozess stellt die *Hüllgeometrie* eines Bauteils eine sinnvolle Vorabversion dar, in der ein Konstrukteur den groben Platzbedarf des zu konstruierenden Bauteils sowie dessen Schnittstellen nach außen spezifiziert. Dieses Datenobjekt erzeugt er als 3D-CAD-Modell am Anfang einer neuen Konstruktionstätigkeit, um danach schrittweise seine Konstruktion zu verfeinern.

Auf der Basis der Hüllgeometrie kann der im Prozess nachfolgende DMU-Prüfer bereits erste Konflikte zwischen den Bauteilen erkennen, während die Bauteil-Konstrukteure ihre Entwicklung noch verfeinern. Durch ein solches *prozesskoordiniertes Simultaneous-Engineering* können Konflikte verschiedener Bauteile früher erkannt und den betroffenen Konstruktionsbereichen mitgeteilt werden, wodurch unnötige Arbeit vermieden wird und damit Zeit und Kosten eingespart werden [Ehr95, Het00]. Darüber hinaus führen diese raschen Rückmeldungen häufig auch zu besseren Konstruktionen, da zu Beginn einer Entwicklungsaufgabe noch mehr Freiheitsgrade beim Lösen von Problemen vorhanden sind als am Ende einer Konstruktion [AH87, Pah99]. Auch lassen sich hier noch die Belange im Prozess nachfolgender Bereiche wie zum Beispiel der Fertigungsplanung [Het00] und Logistik besser berücksichtigen.

*Betrachtet man wieder die schusssichere Tür, so kann ein DMU-Prüfer auf Basis der (vorläufigen) Hüllgeometrien eine erste virtuelle schusssichere Tür zusammenbauen und prüfen. Er stellt dabei fest, dass die vom Fensterheberkonstrukteur vorgesehene Verlängerung der Fensterheberkurbel, um die Bedienkräfte trotz höherem Gewicht der Scheibe gering zu halten, mit dem Sitz kollidieren würde. Der DMU-Prüfer wird sich mit dem betroffenen Fensterheber-Konstrukteur in Verbindung setzen, um gemeinsam eine Lösung zu finden.*

Abhängig vom zu lösenden Problem und der Arbeitsorte der betroffenen Personen erfolgt ein solcher *ad hoc* initiierter *Abstimmungsprozess* in Form persönlicher Gespräche, über Email oder nach einem vorgegebenen Prozess („Dienstweg“). Eine geeignete IT-Unterstützung muss deshalb dafür synchrone und asynchrone Interaktionsformen bereitstellen.

In der Praxis fehlt es jedoch an einer systemtechnischen Umsetzung solcher *prozesskoordinierten Simultaneous-Engineering-Phasen* [Het00]. Zum einen ist ein *nahtloser und integrierter* Übergang zwischen synchronen und asynchronen Interaktionsformen systemtechnisch nicht möglich. Zum anderen wird weder die Weitergabe vorläufiger Daten noch das „Nachversorgen“ fehlender, korrigierter und/oder endgültiger Daten mit bisherigen Systemen unterstützt. Insbesondere der automatischen Nachversorgung kommt in der Praxis eine große Bedeutung zu, um inkonsistente Daten aufgrund unterschiedlicher Aktualitätsstände zu verhindern [Sof01].

*Fensterheberkonstrukteur und DMU-Prüfer kommen überein, dass eine vernünftige Lösung mit einem mechanischen Fensterheber nicht realisiert werden kann. Eine*

*Alternative stellt der Einsatz eines* elektrischen *Fensterhebers dar, so dass man beschließt, diesen Lösungsweg zu verfolgen.*

*Der elektrische Fensterheber wird im Gegensatz zum manuellen Fensterheber von einem externen Konstruktionspartner entwickelt. Vor Start der neuen Entwicklungsiterationsschleife muss der bisherige Änderungsauftrag deshalb um den mechanischen Fensterheber reduziert und durch den elektrischen Fensterheber ergänzt werden. Außerdem muss der Elektrikbereich in den Änderungsauftrag mit aufgenommen werden, um die elektrischen Verbindungen zum Fensterheber zu überprüfen und anzupassen.*

*In der neuen Iterationsschleife existieren damit vier parallele DesignPart-Prozessschritte: die bisherigen Schritte für die Entwicklung der Scheibe und des Türrahmens sowie zwei neue Entwicklungsaufträge für den elektrischen Fensterheber und die Elektrik. Der beauftragte Prozessschritt für die mechanische Fensterheberentwicklung muss dagegen zurückgezogen werden. Dazu muss der Bereich für die mechanische Fensterheberentwicklung informiert werden. Außerdem müssen die in diesem Prozessschritt bereits erzeugten Daten automatisch als nicht mehr relevant markiert und gegebenenfalls gelöscht werden.*

**Dynamische Anpassung einer vorgegebenen Prozessstruktur**

An diesem einfachen Szenario wird schon deutlich, dass Entwicklungsprozesse *variable* parallele Prozessabschnitte besitzen, deren Ausprägung durch die gegenwärtige Objektstruktur, wie beispielsweise der Umfang eines Änderungsauftrags, bestimmt wird. Eine *dynamische* und möglichst *automatische* Anpassung der vorgegebenen Prozessstruktur gemäß der aktuellen Produktausprägung ist deshalb für eine automatisierte Prozesssteuerung unerlässlich. Eine *manuelle* Umgestaltung des Prozesses bei zum Beispiel jeder Veränderung eines Änderungsauftrags ist nicht praxisgerecht. Wichtig bei der automatischen Umgestaltung laufender Prozesse ist auch die Sicherung der Prozesskonsistenz. Nicht mehr benötigte Zweige müssen geeignet zurückgesetzt, weiterhin notwendige Zweige mit neuesten Daten (zum Beispiel aktualisierter Änderungsumfang) versorgt werden.

**Abstimmung Prozesssteuerung mit adaptiver Projektplanung**

Alle Entwicklungsprozesse sind in einem übergeordneten *Fahrzeugprojektplan* eingeordnet [Ehr95, NW98]. Dieser Projektplan dient der Planung und dem Controlling der gesamten Fahrzeugentwicklung. Dabei werden unter anderem auch zeitliche Vorgaben für verschiedene Entwicklungsphasen gesetzt. Diese stellen verbindliche Meilensteine für die operativen Prozesse wie beispielsweise den soeben skizzierten Änderungsprozess dar.

Verletzungen dieser Zeitrestriktionen müssen frühzeitig sowie möglichst systemunterstützt erkannt und unverzüglich der Projektplanung mitgeteilt werden.

Bei der üblichen Laufzeit von Fahrzeugprojekten ist es unvermeidlich, dass sich Projektplanungsphase und -ausführungsphase zeitlich überlappen [Gol96, Mau96, SHL+98]. Eine solche schrittweise Planung führt zusätzlich zu den durch äußere Einflüsse bedingten Planungsveränderungen zu Verschiebungen der Meilensteinvorgaben. Projektplanern fehlen bei der heutigen IT-Landschaft häufig aktuelle Informationen über den gegenwärtigen

Bearbeitungsstatus laufender Entwicklungsprozesse. Sie können damit nur unzureichend abschätzen, wie sich ihre Planungsveränderungen auf die operativen Prozesse auswirken. Insbesondere das Vorverlegen von Zeitrestriktionen führt häufig zu Mehrkosten durch Mehrarbeit oder zu einem nur unzureichenden Erkennen von Meilensteinverletzungen. Auch ist für Bearbeiter einzelner Prozessschritte häufig nicht nachvollziehbar, welche größeren Auswirkungen auf den Gesamtprozess das Nichteinhalten ihrer zeitlichen Vorgaben verursacht. Eine größere Transparenz wird von allen Projektbeteiligten gefordert.

**Effiziente Informationslogistik**

Schon anhand dieses einfachen Beispiels wurde deutlich, dass für jeden Entwicklungsprozessschritt aktuelle und konsistente Informationen möglichst automatisch bereitgestellt werden müssen.[1] Aufgrund der während einer Produktentwicklung großen Menge an anfallenden Daten ist eine *effiziente Informationslogistik* unverzichtbar. Darunter versteht man, dass ein Bearbeiter nur die für die anstehende Aufgabe unmittelbar benötigten Daten erhalten soll, um nicht in einer Flut irrelevanter Daten unterzugehen. Nur so kann gewährleistet werden, dass er in einer vernetzten digitalen Welt mit seiner Arbeit schnell produktiv beginnen kann. Auch können damit Sicherheitsaspekte insbesondere bei der Zusammenarbeit mit externen Partnern (*need-to-know-Prinzip*) und die technische Bereitstellung der großen Datenmengen in einem weltweiten Entwicklungsverbund leichter realisiert werden [MDEF01].

**Sicherstellung einer korrekten Prozesssteuerung**

Plant ein Unternehmen seine wertschöpfenden Prozesse computerunterstützt zu koordinieren, so überträgt es dem IT-System die Verantwortung für die korrekte Steuerung der Prozesse im Sinne der modellierten Unternehmensvorgaben. Dies wird ein Unternehmen nur dann tun, wenn der IT-Systemlieferant bestimmte dynamische Eigenschaften seines Systems, wie beispielsweise eine konsistente Datenversorgung und eine garantierte Prozessterminierung, zusichern kann. Vertrauensverluste der Kunden eines Unternehmens und enorme finanzielle Kosten sind nur einige der möglichen und unbedingt zu vermeidende Schäden, die einem Unternehmen durch fehlerhafte computergesteuerte Prozesse entstehen können.

**Prozessorientierte Integration der vorhandenen Systemlandschaft**

Ein System zur Prozesssteuerung ist eingebettet zwischen einer Vielzahl von Anwendungssystemen (beispielsweise CAD-Werkzeuge für Konstrukteure) sowie Informationssystemen, welche die Daten- und Versionsverwaltung der erzeugten Ergebnisse in der Produktentwicklung übernehmen. Eine effiziente Prozesskoordination muss auch für ein integratives Zusammenspiel der verschiedenen Komponenten sorgen. Nur so lässt sich ein Mehraufwand für die Endanwender durch zum Beispiel wiederholtes Erfassen gleicher Informationen vermeiden.

[1] Studien bei Mercedes-Benz und Opel haben aufgedeckt, dass in Produktionsplanungsbereichen 60% der Arbeitszeit für die Informationssuche und -synchronisation aufgewandt wird [Hal01].

Für den Einsatz eines Prozesssteuerungssystems in einem großen Konzern sind außerdem dessen IT-Strategien zu berücksichtigen. Eine offene, komponentenbasierte und an Standards orientierte Systemarchitektur bietet hier die besten Möglichkeiten für die Integration vorhandener und zukünftiger Systeme.

# Kapitel 3

# Ziel und Aufbau der Arbeit

## 3.1 Zielsetzung

Das einfache Beispiel eines Produktentwicklungsprozesses aus dem vorherigen Kapitel zeigt bereits, dass eine Verbesserung der Produktentwicklungsprozesse immer arbeitsorganisatorische und informationstechnologische Facetten besitzt. Diese Arbeit konzentriert sich auf die informationstechnologische Ebene. Manche der skizzierten Anforderungen werden bereits aus Sicht der Informationstechnologie wissenschaftlich bearbeitet und zum Teil liegen schon Ergebnisse vor, die jedoch im Wesentlichen den statischen Aspekt (Datenintegration und -management, beispielsweise [Sau98, KRS98], Wissensverarbeitung, beispielsweise [KS95]) behandeln. Die prozessorientierten Fragestellungen wurden dagegen bisher nur unzureichend geklärt. Für das anvisierte Einsatzgebiet der Produktentwicklung gilt dies auch für die Workflow-Management-Technologie. So haben Analysen gezeigt, dass die bisherigen Workflow-Management-Konzepte noch erhebliche Defizite bei der Unterstützung von Entwicklungsprozessen aufweisen [BD96a, BKJ+99, HB97, RM97, NW98]. Greift man die im vorherigen Kapitel erwähnten Anforderungen auf, so sind insbesondere noch die folgenden Problemstellungen aus informationstechnologischer Sicht ungelöst, für die im Rahmen dieser Arbeit praxisgerechte Lösungskonzepte erarbeitet werden sollen:

### 3.1.1 Unterstützung unstrukturierter Teilprozesse innerhalb vorgegebener Prozessstrukturen

Die Festlegung einer starren Kontroll- und Datenflussreihenfolge behindert die Kreativität der Bearbeiter unnötigerweise [Pah99, NW98]. Diese in heutigen Workflow-Management-Systemen übliche prozessorientierte Modellierung („*Wie*, also in welchen Prozessschritten, ist die Aufgabe zu tun?“) ist für kreative Aufgaben mit wechselnden Problemstellungen nicht geeignet, da jede Reihenfolgevorgabe die Arbeitsweise nicht richtig darstellt und deshalb auf einen Bearbeiter unangemessen wirkt. Zielorientierte Modellierungskonzepte („*Was* muss als Ergebnis der Aufgabe vorliegen?“) aus dem Bereich der regelbasierten künstlichen Intelligenz [Gro92, GGHV95] oder Groupware-Ansätze [DCSCW98, RSVW94] sind hier einer prozessorientierten Beschreibung vorzuziehen [Bre95, Sac95, HB97, NW98].

Anderseits kann die Beschreibung des Gesamtprozesses nur prozessorientiert durch Verknüpfung der Prozessschritte mittels Kontroll- und Datenfluss erfolgen, um die Prozesskomplexität in der Praxis beherrschen zu können.

Die Herausforderung besteht also in der Integration prozess- und zielorientierter Konzepte [EWM+98]. Statt sich – wie bei bisherigen Ansätzen üblich – als konträre Konzepte gegenseitig auszuschließen, müssen sie sich ergänzen und einen nahtlosen Übergang in die jeweils „andere Welt“ erlauben. Nur so lässt sich eine vorzeitige Datenweitergabe während der Bearbeitung eines unstrukturierten Subprozesses an Folgeschritte im übergeordneten strukturierten Gesamtprozess realisieren.

### 3.1.2 Prozesskoordiniertes Simultaneous-Engineering

#### Vorzeitige Datenweitergabe

Will man vorzeitige Datenweitergabe computergestützt ermöglichen, so muss bei der Datenmodellierung innerhalb eines Workflows der „Grad der Vorläufigkeit“ festgelegt werden: Es muss zum einen modelliert werden, welche minimale und maximale *Datenqualität* ein Eingabeparameter eines Prozessschritts benötigt, damit er sinnvoll bearbeitet werden kann. Zum anderen muss auch definiert werden, in welchen Qualitätsstufen ein Prozessschritt vorläufige und endgültige Ausgabedaten liefern wird. Aufgrund der Vorläufigkeit der Daten müssen zusätzliche Regeln bei der Modellierung der Datenflüsse zwischen den Prozessschritten beachtet werden. Es besteht sonst die Gefahr, dass ein Prozessschritt nicht bearbeitet werden kann, da zwar alle Eingabeparameter mit typkonformen aktuellen Daten versorgt sind, diese jedoch nicht in der richtigen Qualität vorliegen.

Vorzeitige Datenweitergabe bedingt auch neue Benutzerinteraktionen zur Laufzeit. Bei den bisherigen Workflow-Management-Ansätzen erfolgt die Weitergabe der Daten implizit durch Beenden eines Prozessschritts. Sollen jedoch noch während der Schrittbearbeitung Daten weitergegeben werden, damit Folgeschritte hiermit bereits arbeiten können, so müssen *Datenweitergabe* und *Schrittbeendigung* zwei getrennte Operationen darstellen. Der Aufruf der Datenweitergabeoperation beim Erreichen einer modellierten Qualitätsstufe kann automatisch durch das Workflow-Management-System erfolgen oder muss explizit durch den Schrittbearbeiter angestoßen werden. Da in der Regel die Verantwortung für die Datenqualität bei der bearbeitenden Person liegt und eine völlig automatische Überprüfung der Datenqualität häufig nicht möglich ist, ist die „unbemerkte“ Weitergabe noch vorläufiger Daten nicht sinnvoll.

Vorzeitige Datenweitergabe erschwert die Sicherung der Datenkonsistenz. Notwendig sind Konzepte, die eingetroffene aktuelle Daten mit den bereits vorliegenden alten Eingabedaten abgleichen. Ansätze aus dem Bereich des Versions- und Replikaktionsmanagements [KRS98, BK86, Kat90, BD96b] müssen für den Datenfluss von Workflows geeignet adaptiert werden.

Die Datenkonsistenz kann auch durch zyklische Weitergabe vorläufiger Daten gefährdet werden. Solche Datenflusszyklen, die bei Schleifen und durch Datenflussabhängigkeiten zwischen parallelen Zweigen auftreten können, müssen zur Laufzeit eines Workflows zumindest erkannt und geeignet blockiert werden. Besser ist es allerdings durch Modellierungsregeln oder -analysen diese Gefahren vorbeugend, also noch vor der Workflow-Ausführung, auszuschließen.

**Bereitstellung von Ad-hoc-Interaktionsformen**

Die Herausforderung besteht hier weniger in der Entwicklung neuer Interaktionsformen, sondern in der Integration (synchroner) Groupware-Konzepte in eine vorgegebene asynchrone Workflow-Steuerung. Statt eines „unwissenden“ Nebeneinanders [WPSH+96, AGSC95] von Groupware und Workflow-Koordination muss bei der ad hoc Initiierung einer Abstimmungsrunde der Kontext des übergeordneten Workflows zur Verfügung stehen [EWM+98]. Damit lassen sich automatisch die betroffenen Personen und benötigte Daten anhand des Workflow-Zustands ableiten und Ergebnisse in den weiteren Workflow-Ablauf integrieren. Somit wird der administrative Zusatzaufwand für die Prozessbeteiligten minimiert.

### 3.1.3 Dynamische Anpassung einer vorgegebenen Prozessstruktur

Der Schwerpunkt bei dynamischen Anpassungen einer vorgegebenen Prozessstruktur liegt sicherlich in der Frage der Automatisierung der Workflow-Umgestaltung [Kra97]. Bisherige Ansätze aus dem Bereich des flexiblen Workflow-Managements (siehe zum Beispiel [Rei00]) konzentrieren sich auf benutzerinitiierte Abweichungen von einer vorgegebenen Workflow-Struktur.

Dagegen muss für Produktentwicklungsprozesse der Umbau automatisch erfolgen, indem durch Analyse von im Workflow erzeugten Objektstrukturen, wie beispielsweise die Objektstruktur eines Änderungsauftrags, die neue Workflow-Struktur berechnet wird. Dies erfordert ein enges Zusammenspiel zwischen Workflow-Steuerung und dem Datenmanagementsystem [Kat90], in dem die strukturierten Objekte verwaltet werden. Durch die mögliche Vorläufigkeit der Objektversionen muss dabei eine bereits durchgeführte Umgestaltung eines Workflows aufgrund neu eingetroffener Daten rückgängig gemacht werden und entsprechend der Struktur der aktuelleren Daten erneut angestoßen werden. Natürlich dürfen wichtige dynamische Eigenschaften wie Blockierungsfreiheit, Terminierungszusicherung oder Bewahrung der Datenkonsistenz durch die Anpassung der Workflow-Struktur nicht gefährdet werden.

### 3.1.4 Abstimmung Prozesssteuerung mit adaptiver Projektplanung

Bisherige Workflow-Management-Konzepte gehen davon aus, dass die Workflow-Steuerung nur interne Regeln (modelliert im Wesentlichen durch Kontroll- und Datenfluss) beachten muss. Externe Regeln, insbesondere Restriktionen durch übergeordnete Meilensteine aus einem Projektplanungssystem, wurden bisher nicht hinreichend berücksichtigt. Die folgenden in der Praxis wichtigen Fragen sind hier zu klären:

- Wie kann das entwickelte Workflow-Management-Modell in einem projektorientierten Gesamtkonzept integriert werden?
- Welche Auswirkungen hat die notwendige zeitliche Überlappung zwischen adaptiver Projektplanung und Workflow-Steuerung?
- Was sind sinnvolle Schnittstellen zwischen Projektplanung und Workflow-Steuerung?
- Wie wird auf Veränderungen in der jeweils anderen Ebene reagiert?

### 3.1.5 Sicherstellung einer korrekten Prozesssteuerung

Eine korrekte Prozesssteuerung umfasst sicher die Aspekte der konsistenten Datenversorgung, der Blockierungsfreiheit (Deadlocks, Lifelocks) und der garantierten Workflow-Terminierung. Der Nachweis dieser dynamischen Eigenschaften kann auf verschiedenen Ebenen (Workflow-Engine, Workflow-Modell, Workflow-Instanz) und auf verschiedene Arten (Beweis, Analyse, Simulation, Animation) erfolgen. Nachteilig bei Simulation und Animation ist, dass die geforderten Eigenschaften nur für untersuchten Simulations- beziehungsweise Animationsfälle garantiert werden können. Für nicht untersuchte Fälle kann eine korrekte Prozesssteuerung zur Laufzeit nicht garantiert werden. „Unliebsame Überraschungen“ können nicht ausgeschlossen werden. Beweis- und Analysemethoden [Aal97, WWD+97, HOR96] sind deshalb Simulations- und Animationsverfahren [Obe94a] vorzuziehen.

Wichtig hierbei ist, dass die spezifischen Anforderungen (vorzeitige Datenweitergabe für Simultaneous-Engineering, Schleifen, Datenabhängigkeiten zwischen parallelen Zweigen) entsprechend beim Nachweis der dynamischen Eigenschaften berücksichtigt werden.

## 3.2 Schlussfolgerungen und weiterer Aufbau der Arbeit

Bei diesen Fragestellungen handelt es sich nicht primär um ein Problem der technischen Umsetzbarkeit. Vielmehr besteht hier noch grundsätzlicher konzeptioneller Klärungsbedarf [NW98]. Die Entwicklung *innovativer Workflow-Modellierungs- und Ausführungskonzepte* für eine praxisgerechte IT-Koordination von Produktentwicklungsprozessen ist das erklärte Ziel dieser Arbeit. Das dabei entwickelte **WEP**-Workflow-Management-System (**WEP** = Workflow Management for Engineering Processes) soll das Zusammenspiel der in ihrer Gesamtheit neuartigen Konzepte demonstrieren. Die Abbildung 3.1 zeigt im Überblick die im Rahmen dieser Arbeit entwickelten Konzepte und Lösungsansätze und ihre Zuordnung zu den anvisierten Anforderungen. Dieser Überblick macht bereits zwei Punkte deutlich:

1. Einerseits können manche der **WEP**-Konzepte zur Realisierung gleich mehrerer Anwenderanforderungen eingesetzt werden. Diese Konzepte erkennt man in der Abbildung an ihrem Farbverlauf.

2. Anderseits können einige Anforderungen nur durch die Kombination mehrerer **WEP**-Konzepte adäquat umgesetzt werden. Das ergibt sich häufig schon aus der Tatsache, dass diese Anforderungen neue Konzepte für die Modellierung *und* Ausführung von Workflows benötigen.

Die in der Abbildung aufgeführten **WEP**-Konzepte enthalten außerdem Verweise auf das jeweilige Kapitel in der Arbeit, in dem das Konzept erläutert wird. Damit kann der Leser schnell das richtige Kapitel finden, wenn er sich nur für die Umsetzung bestimmter Anforderungen interessiert.[1]

Der Leser wird auch erkennen, dass nicht alle Problemstellungen in gleicher Ausführlichkeit in der Arbeit behandelt werden, da dies den Umfang der Arbeit sprengen würde. Teilweise wird auch auf Diplomarbeiten [Sau99, Kno99, Sch00, Kos00] verwiesen, die begleitend zu dieser Dissertation entstanden sind und bei denen Teilfragestellung in Zusammenarbeit mit dem Autor detaillierter untersucht wurden.

[1] Die in der Abbildung nicht erwähnten Kapitel sind in der Regel einleitende Kapitel.

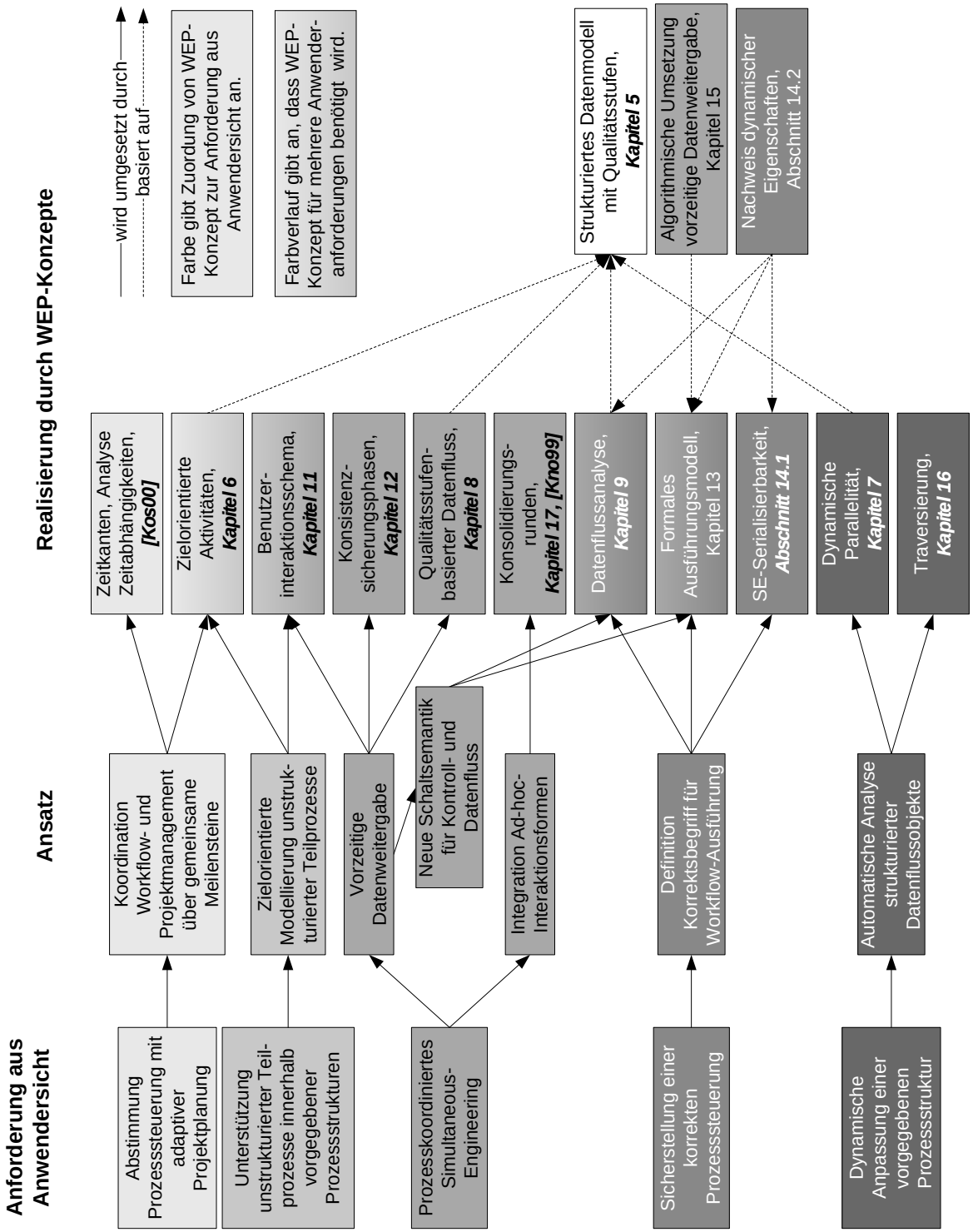



**Abbildung 3.1:** Zuordnung der Anforderungen auf die **WEP**-Konzepte

Die weitere Arbeit setzt sich damit aus den folgenden Teilen zusammen:

Im Teil II (**WEP**-Workflow-Modellierung) und Teil III (**WEP**-Workflow-Ausführung) sind alle neu entwickelten **WEP**-Konzepte enthalten, die bereits in der Abbildung 3.1 erwähnt wurden.

Im Teil IV werden die entwickelten **WEP**-Konzepte mit anderen Ansätzen aus der Literatur und Praxis verglichen und entsprechend eingeordnet. Kriterien für den Vergleich sind die in Teil I beschriebenen Anforderungen aus Anwendersicht.

Teil V fasst die Hauptergebnisse zusammen und gibt einen Ausblick auf weiterführende Themen. Hier wird unter anderem aufgezeigt, dass die entwickelten **WEP**-Konzepte – trotz ihrer primären Orientierung an den Anforderungen aus der Fahrzeugentwicklung – so generisch sind, dass sie auch für andere Anwendungsgebiete mit semi-strukturierten Prozessen geeignet sind.

# Teil II

# Das WEP-Modell: Adäquate Modellierung von Entwicklungsprozessen

# Kapitel 4

# Einführung

Teil II der Arbeit widmet sich der Entwicklung geeigneter Modellierungskonzepte, um Produktentwicklungsprozesse mit den in Teil I gefundenen Anforderungen adäquat in Form von ***WEP**-Workflows* beschreiben zu können. Die Modellierung eines **WEP**-Workflows muss dabei alle Aspekte umfassen, um zur Laufzeit (siehe Teil III) eine korrekte Prozesssteuerung gewährleisten zu können, inklusive der Koordination unstrukturierter Teilprozesse und Simultaneous-Engineering-Phasen, des automatisierten Umbaus von Workflow-Strukturen und der Integration übergeordneter Projektplanungen.

## 4.1 Entwurfsziele beim WEP-Modell

Die wesentlichen Zielsetzungen bei der Entwicklung des **WEP**-Modells sind:

*Orthogonalität:*
: Die erweiterten Modellierungskonzepte müssen in beliebiger Weise kombinierbar sein. Nur damit lässt sich Wiederverwendbarkeit realisieren, die den Aufbau komplexerer Prozesse durch Schachtelung ermöglicht.

*Einbettung unstrukturierter Teilprozesse:*
: Das **WEP**-Modell muss Modellierungskonstrukte für eine nahtlose Einbettung unstrukturierter Teilprozesse bereitstellen.

*Unterstützung externer, komplex strukturierter Datenmodelle:*
: Die in den Prozessmodellen verwendeten Datenobjekte sind durch hohe Komplexität und Vernetzung gekennzeichnet. In der Regel sind diese Modelle bereits vorgegeben und müssen bei der Prozessgestaltung verwendet werden.

*Datenqualitäten:*
: Produktentwicklungsprozesse werden auch durch die Festlegung von semantischen Informationen über die zu transferierenden Datenobjekte in Form von Datenqualitäten modelliert. Das **WEP**-Modell muss das Importieren bereits vorhandener Beschreibungen von Datenqualitäten und gleichzeitig ein generisches Erweitern ermöglichen.

*Modellierung vorzeitiger Datenweitergabe:*
: Die vorzeitige Weitergabe von Datenobjekten zur Laufzeit kann nur dann konsistent erfolgen, wenn die Weitergabebedingungen zur Modellierungszeit geeignet festgelegt werden.

Hierfür sind die notwendigen Modellierungskonstrukte zu definieren und semantische Modellierungsregeln festzulegen.

*Modellierung von Zielen und Zeitaspekten:*
Wie im ersten Teil der Arbeit gezeigt, lassen sich unstrukturierte Teilprozesse am besten durch Angabe von Zielen beschreiben. Unabdingbar für eine systemseitige Überwachung der Ziele sind Vorgaben, wann die Ziele zeitlich erreicht werden müssen.

*Modellierung dynamischer Parallelität:*
Da die genaue Ausprägung eines semi-strukturierten Prozesses von der augenblicklichen Ausprägung der Datenobjekte abhängt, müssen die erlaubten Veränderungen beschrieben werden, damit sie zur Laufzeit automatisch ausgeführt werden können.

*Formalisierbarkeit:*
Die für den Workflow-Modellierer zur Modellierung von Produktentwicklungsprozessen bereitgestellte abstrakte Workflow-Beschreibungssprache muss auf einem formalen Beschreibung- und Ausführungsmodell beruhen. Nur so können zur Modellierungszeit präzise Aussagen über das spätere Ausführungsverhalten des Workflows getroffen werden. Dazu ist es notwendig, zur Modellierungszeit potenzielle Fehler im Entwurf eines Workflows durch Einhaltung geeigneter Konstruktionsprinzipien beziehungsweise durch Festlegung formaler und überprüfbarer Korrektheitseigenschaften zu erkennen und auszuschließen. Dies ist insbesondere bei einem Workflow-Modell wie dem **WEP**-Modell wichtig, das eine weitaus größere Modellierungsmächtigkeit und Ausführungsfunktionalität bereitstellt als herkömmliche Workflow-Management-Systeme. Beispiele für zu überprüfende Korrektheitsbedingungen sind korrekte Versorgung mit Aufrufparametern oder Aussagen über das dynamische Verhalten eines Workflows wie das Erreichen eines wohldefinierten Endzustands.

*Generisches Modell:*
Das zu entwickelnde Modell muss generisch sein, damit es für beliebige Anwendungsgebiete semi-strukturierter Prozesse einsetzbar ist.

## 4.2 Grundlagen des WEP-Modells

Bis zu einem gewissen Detaillierungsgrad findet man auch bei Entwicklungsprozessen sich wiederholende Vorgehensmuster [KK94, Ehr95, VFS97], so dass eine prozessorientierte Modellierung sinnvoll und häufig auch durch Unternehmensrichtlinien oder Verfahrensanweisungen [VDI86] vorgeschrieben ist. Beispielsweise muss in Abbildung 2.2 ein Änderungsprozess aus den Schritten *DefineChangeRequest — DesignPart — CheckDMU — . . .* bestehen.

Will man jedoch in gleicher Weise die aufgrund ihres individuellen und iterativen Charakters wenig vorstrukturierten Entwicklungskernprozesse (z. B.: *DesignPart*) modellieren, so stößt der prozessorientierte Ansatz an seine Grenzen und bietet deshalb nur unzureichende Unterstützung bei den im vorherigen Teil der Arbeit skizzierten Anforderungen.

Das nun im Folgenden vorgestellte **WEP**-Modell beschreibt solche semi-strukturierten Prozesse nur im „Groben“ mittels *prozessorientierter Methodiken* und ermöglicht damit eine Führung der am Prozess beteiligten Mitarbeiter entsprechend den Unternehmensrichtlinien.

Bei schwach strukturierten Teilprozessen erfolgt ein Paradigmenwechsel von *prozessorientierter* zu *zielorientierter* Modellierung: Auf eine aufwändige und wenig nützliche prozessorientierte Reihen-

folgemodellierung der Subprozesse innerhalb eines schwach strukturierten Prozesses wird hierbei verzichtet. Der gesamte unstrukturierte Teilprozess wird zu einer sogenannten *zielorientierten Aktivität* zusammengefasst. Den Bearbeitern werden statt Reihenfolgevorgaben detaillierte Angaben gemacht, zu *welchen Zeitpunkten* sie *welche Ergebnisobjekte* in *welcher Datenqualität* abzuliefern haben (= *Ziel des Prozesses*, siehe Kapitel 6). Dazu ist es allerdings notwendig, Datenqualitäten (siehe Kapitel 5) und Zeitrestriktionen (siehe Abschnitt 6.2.2.2) zu modellieren.

Damit ist eine Voraussetzung geschaffen, dass – im Gegensatz zu den Bausteinen bei anderen prozessorientierten Ansätzen – eine zielorientierte Aktivität *vor ihrer Beendigung* Daten an Nachfolgerschritte weitergeben kann.

Zielorientierte Aktivitäten stellen zwar mehr Funktionalität als „übliche“ Prozessbausteine in einem herkömmlichen prozessorientierten Workflow-Management-System zur Verfügung. Sie können trotzdem in analoger Weise zu einem prozessorientierten Gesamtprozess verknüpft werden. Die Verknüpfungskonstrukte des **WEP**-Modells werden in den Kapiteln 7 (Kontrollfluss) und 8 (Datenfluss) sowie im Abschnitt 6.2.2.2 (Zeitrestriktionen) aufgezeigt.

Wiederverwendung und Validierung von Workflows erfordern eine blockorientierte Modellierung. **WEP**-Workflows werden deshalb bezüglich des Kontrollflusses mittels Blöcken aufgebaut, die entweder disjunkt oder ineinander geschachtelt sind.

## 4.3 Festlegung einer formalen Darstellungsform für WEP-Workflows

Ein wichtiger Grund für die Attraktivität des Workflow-Ansatzes ist es, dass Workflows anwendungsnah und auf semantisch hohen Niveau – meist auf grafischer Ebene – beschreibbar sind. Neben der praktischen Anwendbarkeit (hohe adäquate Ausdrucksmächtigkeit für die anvisierten Anwendungsgebiete, kompakte und leicht verständliche Darstellung) ist es jedoch essentiell, dass das Workflow-Modell auf einer formalen Beschreibung basiert.

Nur eine formale Repräsentation erlaubt statische Analysen und Verifikationen von Workflow-Modellen, um Modellierungsfehler frühzeitig aufzudecken beziehungsweise zu vermeiden [Aal97, HOR96]. Auch sind hiermit präzise Aussagen über das zukünftige Systemverhalten sowie über die Korrektheit der Workflow-Modelle realisierbar.

Die formale Repräsentation der **WEP**-Workflow-Modelle wird mittels Grammatikregeln definiert, die neben einer anschaulichen Beschreibung in den folgenden Abschnitten eingeführt werden, in denen die **WEP**-Modellierungskonstrukte vorgestellt werden. In dieser formalen Repräsentation wird ein **WEP**-Workflow als geschachteltes Tupel beschrieben, dessen einzelne Komponenten wiederum Tupel oder Mengen von Tupeln sein können. Diese formale Darstellungsform wurde so gewählt (*LR(1)-Grammatik* [ASU85, HMU00]), dass sie sich einfach und automatisch in andere Sprachen, wie beispielsweise in eine XML-basierte Sprache überführen lässt, auf der die prototypische Implementierung der **WEP**-Konzepte aufsetzt.

# Kapitel 5

# Datenmanagement: Das WEP-Metadatenmodell

## 5.1 Begründung für ein Metadatenmodell

Workflow-Management-Systeme koordinieren auch die Datenflüsse zwischen den einzelnen Prozessschritten. Um jedem Prozessbeteiligten die notwendigen Daten zur Bearbeitung eines Prozessschrittes anzubieten, müssen diese Datenaspekte natürlich auch modelliert werden. Die Grundlage für jede Datenflussmodellierung stellt ein Datenmodell zur Verfügung, auf deren Basis die benötigten Ein- und Ausgabedaten eines Prozessschrittes und deren Verknüpfung über Datenflusskanten definiert werden.

Allerdings bieten Workflow-Management-Systeme als generische Systeme kein Datenmodell an, da das konkrete Datenmodell – wie auch die darauf basierenden Workflows – durch das jeweilige Anwendungsgebiet vorgegeben wird. Stattdessen legen sie mittels *Metadatenmodellen* „Richtlinien“ fest, *wie* die Daten, die in Workflows erzeugt, gelesen und manipuliert werden, modelliert werden müssen. Da Produktentwicklungsprozesse in der Regel über komplex strukturierte Datenmodelle verfügen, kommt diesem Metadatenmodell besondere Bedeutung zu. Für komplexe Datenstrukturen haben sich objektorientierte Datenmodelle als die zweckmäßigste Darstellungsform zur Datenmodellierung herausgestellt ([RMHN94], vgl. Standard STEP/EXPRESS [MS91, Lüh94, ISO99a, ISO99b] im technischen Entwicklungsbereich). Das Metadatenmodell im **WEP**-Modell fordert deshalb ein objektorientiertes Datenmodell mit Einfachvererbung. Die Objekte besitzen Attribute und Relationen, die Beziehungen zu anderen Objekten beschreiben.

Diese Minimalforderungen werden von vielen Datenmodellen in der Praxis erfüllt, wie beispielsweise die bei technischen Entwicklungen weit verbreiteten *Application Protocols* der STEP-Norm [MS91, Lüh94, ISO99a, ISO99b]. Dies gilt ebenso für Produktdatenmanagementsysteme ([Kat90], **Metaphase** [SDRC99], **VPM** [IBM99a],**Enovia R5** [DS02], **Windchill** [PTC01b], **eMatrix** [Matrix01a], PDM-Enabler [OMG00]).

Der Import bereits vorhandener Datenmodelle ist notwendig, da eine manuelle Nachmodellierung bei der vorhandenen Komplexität aufwändig, fehleranfällig und nicht praxisgerecht ist.

Im weiteren Teil der Arbeit wird mit $OCLASS$ die Menge der Objektklassen des konkret verwendeten Datenmodells bezeichnet, das den Konventionen des **WEP**-Metadatenmodells entspricht.

Dagegen sind Beschreibungen semantischer Informationen, wie die Datenqualität von Objekten meist nicht vorhanden [Kat90, BS01]. Für die geforderte vorzeitige Datenweitergabe sind solche semantische Informationen unabdingbar. Der praktische Einsatz des **WEP**-Modells setzt voraus, dass die Beschreibung der semantischen Informationen möglichst generisch sind, um wiederum für beliebige Anwendungsfelder einsetzbar zu sein. Das **WEP**-Metadatenmodell fordert deshalb nur, dass Datenqualitäten über Belegungen der Objektattribute und über die Datenqualitäten der Subobjekte zur Verfügung gestellt werden. Der folgende Abschnitt beschreibt im Detail, wie die Datenqualitäten im **WEP**-Modell zur Verfügung gestellt werden müssen.

## 5.2 Beschreibung von Datenqualitäten

Eine Datenqualitätsstufe (*quality level*) wird mittels prädikatenlogischer Ausdrücke über die Attribute einer Objektklasse und ihren Beziehungen zu anderen Objektklassen spezifiziert. Eine Datenqualitätsstufe ist erreicht, wenn *alle* zu dieser Datenqualitätsstufe spezifizierten prädikatenlogischen Ausdrücke ***true*** ergeben.

Um auf jeden Attributtyp anwendbar zu sein, wird nur eine minimale Menge von Vergleichsoperatoren auf den Attributen definiert.[1]

*Gleichheits-Operator AttrIsEq:*
: *AttrIsEq* überprüft, ob ein Attribut den angegeben Wert besitzt.

*IstDefininiert-Operator AttrIsDef:*
: *AttrIsDef* überprüft, ob dem Attribut bereits ein Wert zugewiesen wurde.

Der einzige Vergleichsoperator bei Beziehungen ist der *Gleichheits-Operator,* der auf Qualitätsstufen der Subobjekte angewandt werden kann.

Basierend auf diesen Grundbausteinen lassen sich mittels der Operatoren $\boldsymbol{AND}, \boldsymbol{OR}$ und $\boldsymbol{NOT}$ komplexere Formeln aufbauen. Im Folgenden wird mittels eines *Prädikatenkalküls* formal die Sprache der erlaubten prädikatenlogischen Ausdrücke definiert:

Sei $O \in OCLASS$ eine freie Variable und $oattr, oql, orel, oval$ Konstanten

$$
\begin{aligned}
AbstrBoolExpr(O) &\rightarrow QuantorExpr(O, orel, oql) \mid \\
&\quad\; AtomicExpr(O, oattr, oval) \mid \\
&\quad\; \boldsymbol{NOT}\, AbstrBoolExpr(O) \mid \\
&\quad\; AbstrBoolExpr(O)\ BinExpr\ AbstrBoolExpr(O) \\
QuantorExpr(O, orel, oql) &\rightarrow \forall \mid \exists P \in IsObject(O, orel) : QualLevEq(P, oql) \\
AtomicExpr(O, oattr, oval) &\rightarrow AttrIsEq(O, oattr, oval) \mid AttrIsDef(O, oattr) \\
BinExpr &\rightarrow \boldsymbol{AND} \mid \boldsymbol{OR}
\end{aligned}
$$

Die Interpretation obiger Formeln ist wie folgt:

$IsObject : OCLASS \times RELATIONNAME \rightarrow \wp(OCLASS)$:
: $IsObject(O, orel) =$ Menge aller Subobjekte, die mit $O$ in Relation $orel$ stehen

[1] Natürlich kann diese Menge für spezielle Attributtypen, wie zum Beispiel total geordnete Typen, erweitert werden.

$QualLevEq : OCLASS \times QUALITYLEVELNAME \rightarrow BOOLEAN:$
$QualLevEq(O, oql) =$ ***true***, wenn das Objekt $O$ die Qualitätsstufe $oql$ erreicht hat, andernfalls ***false***

$AttrIsEq : OCLASS \times ATTRNAME \times ATTRVALUE \rightarrow BOOLEAN:$
$AttrIsEq(O, oattr, oval) =$ ***true***, wenn das Attribut $oattr$ des Objekts $O$ den Wert $oval$ enthält, andernfalls ***false***

$AttrIsDef : OCLASS \times ATTRNAME \rightarrow BOOLEAN:$
$AttrIsDef(O, oattr) =$ ***true***, wenn das Attribut $oattr$ des Objekts $O$ bereits irgendeinen Wert enthält, andernfalls false

Wie später ersichtlich sein wird, müssen die Qualitätsstufen jeder Objektklasse total geordnet sein. Für die Spezifikation einer Qualitätsstufe sind verschiedene Varianten möglich. Die Varianten werden zusammen mit ihren Vor- und Nachteilen diskutiert.

### Variante 1: Spezifikation der minimalen Bedingungen pro Qualitätsstufe

Für jede Qualitätsstufe $oql_i$ werden mittels der prädikatenlogischen Ausdrücke die minimalen Bedingungen definiert, die zum Erreichen von $oql_i$ erfüllt sein müssen. Die Bedingungen verschiedener Qualitätsstufen können sich hierbei beliebig überlappen. Folglich muss ein Objekt der Qualitätsstufe $oql_i$ nicht notwendigerweise die Qualitätsstufe $oql_{i-1}$ erfüllen, da nicht gilt $Bedingungen(oql_{i-1}) \subseteq Bedingungen(oql_i)$.

**Semantik der Reihenfolge der Qualitätsstufen:** Die Reihenfolge der Qualitätsstufen beschreibt den üblichen „Lebenszyklus“, den ein Objekt bei seiner inkrementellen Konkretisierung durchlaufen wird. Betrachtet man in der zeitlichen Historie eines Objekts die Reihenfolge der durchlaufenen Qualitätsstufen, so gilt: $Time(oql_{i-1}) < Time(oql_i)$.

### Variante 2: inkrementelle Spezifikation von Qualitätsstufen

Die Bedingungen der Qualitätsstufe $oql_i$ bauen auf den Bedingungen der Qualitätsstufe $oql_{i-1}$ auf. Damit gilt stets: $Bedingungen(oql_{i-1}) \subseteq Bedingungen(oql_i)$.

### Bewertung und Auswahl:

Vorteil der Variante 2 ist, dass der Modellierungsaufwand im Vergleich zu Variante 1 geringer ist, wenn sich die Bedingungen verschiedener Qualitätsstufen stark überlappen.

Bei Variante 1 lassen sich Qualitätsstufen flexibler beschreiben als bei Variante 2, da nicht immer alle Bedingungen der vorigen Qualitätsstufen erfüllt sein müssen. Ein Beispiel aus den Entwicklungsbereichen ist die Existenz einer aktuellen Bauraumbeschreibung, die nur am Anfang einer Bauteilentwicklung (niedrige Qualitätsstufe eines Bauteils) notwendig ist. Bei höheren Qualitätsstufen wird die Bauraumbeschreibung nicht mehr benötigt, da hier schon die detaillierte Bauteilbeschreibung vorliegt. Da sich solche praxisrelevanten Sachverhalte mit Variante 2 nicht geeignet darstellen lassen, wird beim **WEP**-Modell Variante 1 herangezogen.

Die Menge aller Qualititätsstufen einer beliebigen Objektklasse $OCLASS$ wird mit $OQUALITYLEVELS(OCLASS)$ bezeichnet.

## 5.3 Abschließende Illustration der Datenmodellkonzepte

Zum Abschluss des Kapitels über das **WEP**-Metadatenmodell werden die eingeführten Konzepte anhand eines Beispiels illustriert, das sich am Prozessmodell aus Abbildung 2.2 orientiert. Die Abbildung 5.1 zeigt ein **WEP**-konformes Datenmodell, das einen Ausschnitt des Datenmodells für Änderungsaufträge mit zugehörigem Änderungsumfang beschreibt.

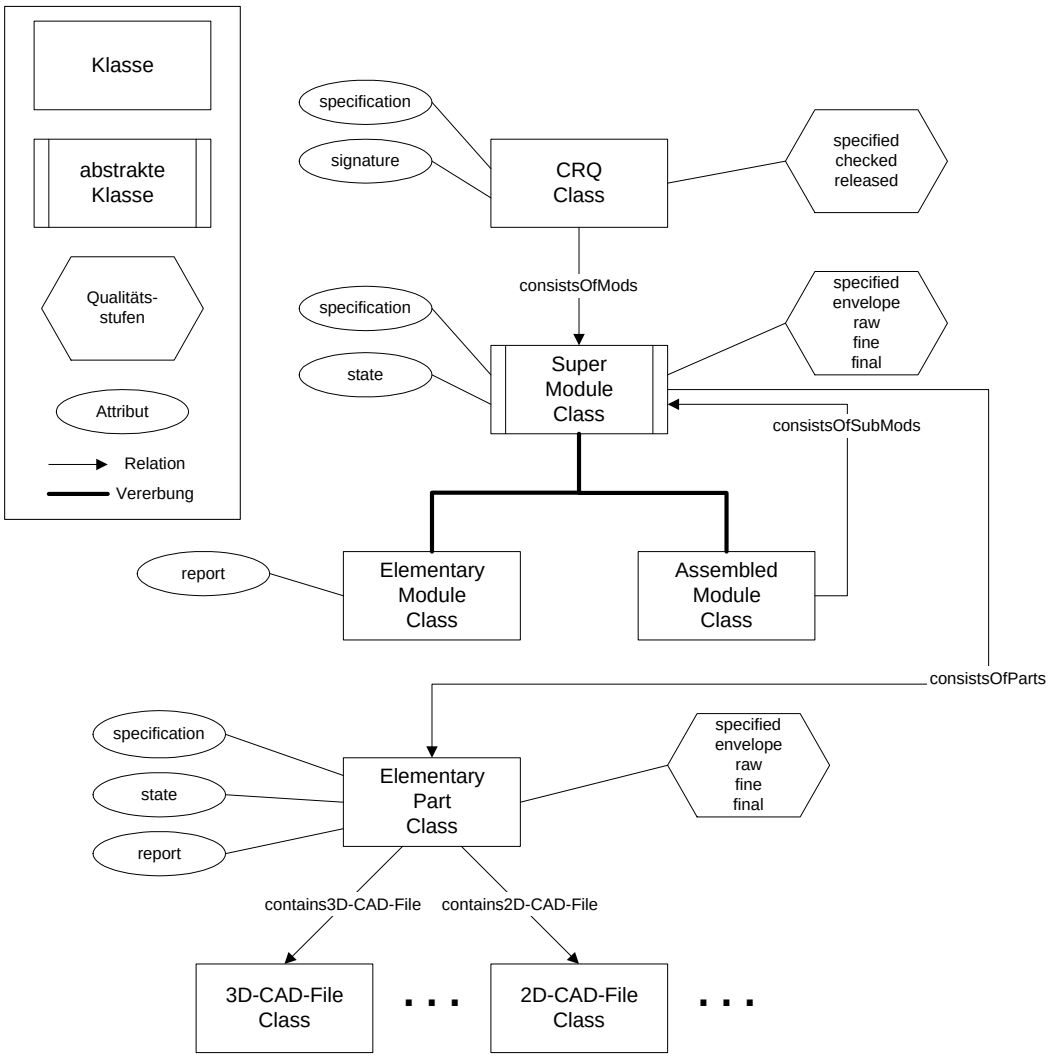



**Abbildung 5.1: WEP**-konformes Datenmodell

Basierend auf diesem Datenmodell werden exemplarisch einige Qualitätsstufen definiert (vergleiche Abbildung 5.2):

$$
\begin{aligned}
CRQClass.specified \quad &:\Leftrightarrow \quad AttrIsDef(CRQClass, specification) \\
&\quad AND \\
&\quad AttrIsDef(CRQClass, signature) \\
CRQClass.checked \quad &:\Leftrightarrow \quad AttrIsDef(CRQClass, signature) \\
&\quad AND \\
&\quad \forall\, mod \in IsObject(CRQClass, conistsOfMods) : \\
&\quad QualityLevEq(mod, raw) \\
Assembled\text{-} & \\
ModuleClass.raw \quad &:\Leftrightarrow \quad AttrIsDef(AssembledModuleClass, specification) \\
&\quad AND \\
&\quad \forall\, submod \in IsObject(AssembledModuleClass, consistsOfSubMods) : \\
&\quad QualityLevEq(submod, raw) \\
Elementary\text{-} & \\
ModuleClass.raw \quad &:\Leftrightarrow \quad AttrIsDef(ElementaryModuleClass, specification) \\
&\quad AND \\
&\quad \forall\, parts \in IsObject(ElementaryModuleClass, consistsOfParts) : \\
&\quad QualityLevEq(parts, raw) \\
PartClass.raw \quad &:\Leftrightarrow \quad AttrIsDef(ElementaryModuleClass, state) \\
&\quad AND \\
&\quad \exists\, cad \in IsObject(PartClass, contains\text{-}3DCAD\text{-}File) : \\
&\quad QualityLevEq(cad, *) \\
PartClass.fine \quad &:\Leftrightarrow \quad AttrIsDef(ElementaryModuleClass, state) \\
&\quad AND \\
&\quad AttrIsDef(ElementaryModuleClass, report) \\
&\quad AND \\
&\quad \exists\, cad \in IsObject(PartClass, contains\text{-}3DCAD\text{-}File) : \\
&\quad QualityLevEq(cad, *)
\end{aligned}
$$

**Abbildung 5.2:** Beispiel für die Definition von Qualitätsstufen für das Datenmodell aus Abbildung 5.1

Zum Erreichen der Qualitätsstufe *checked* muss bei einem Objekt der Klasse *CRQClass* also die Unterschrift (Attribut *signature*) vorhanden sein und jedes über die Relation *conistsOfMods* verknüpfte Modul in der Qualitätsstufe *raw* vorliegen. Abbildung 5.3 zeigt eine Objektinstanz der Klasse *CRQClass* in der Qualitätsstufe *checked*.

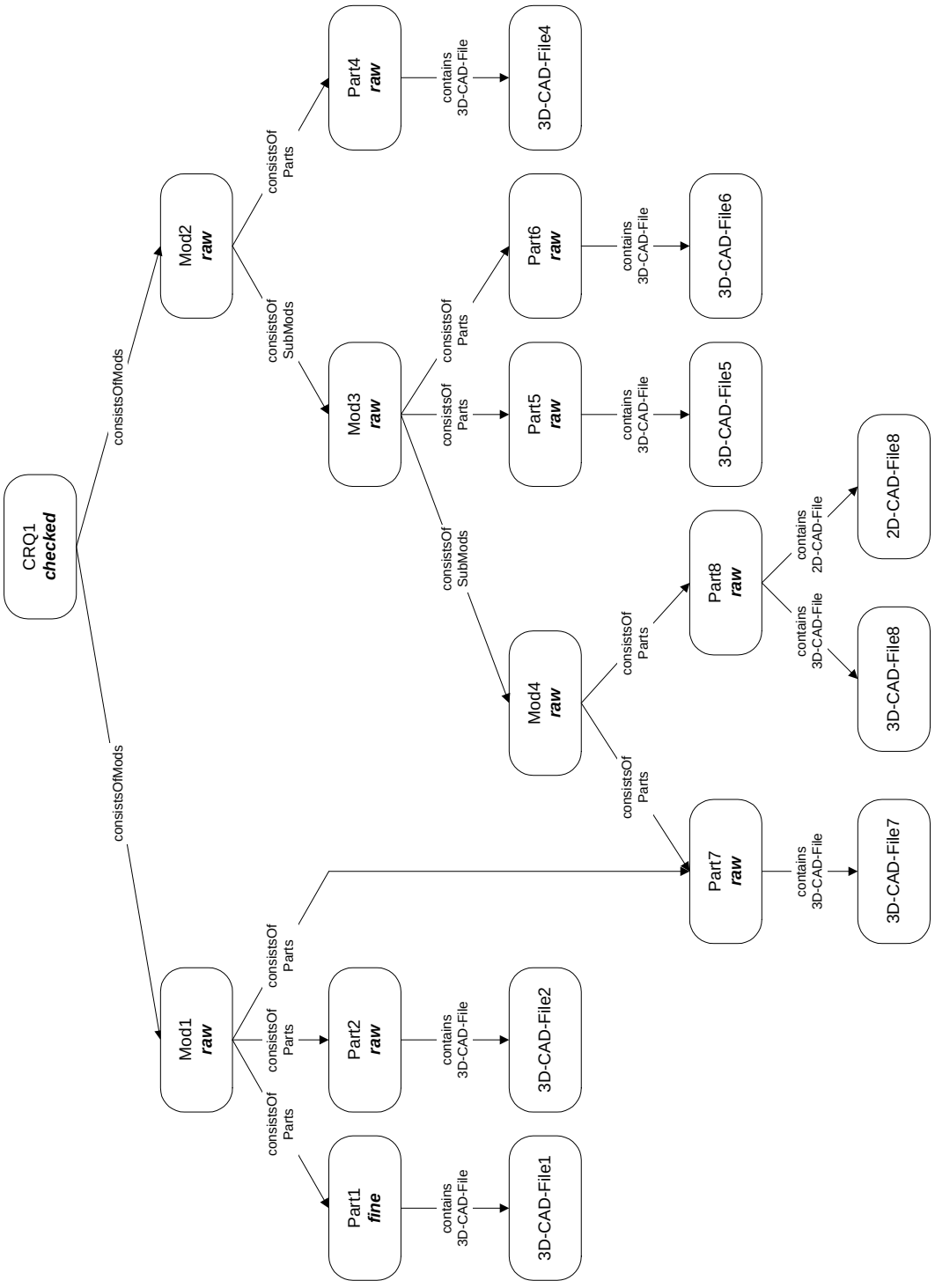



**Abbildung 5.3:** Beispiel einer Objektinstanz zum Datenmodell von Abbildung 5.1

# Kapitel 6

# Behandlung unstrukturierter Teilprozesse

Die Unterstützung unstrukturierter Teilprozesse stellt eine der grundlegenden Anforderungen für ein Workflow-Management-System dar, das semi-strukturierte Prozesse wie Produktentwicklungsprozesse koordinieren soll.

## 6.1 Vorgehensweise

Im **WEP**-Modell erfolgt eine *zielorientierte* Modellierung unstrukturierter Prozesse. Dazu werden alle Subprozessschritte eines unstrukturierten Prozesses zu einer *zielorientierten Aktivität* zusammengefasst. Innerhalb einer zielorientierten Aktivität können alle Subprozessschritte in beliebiger Reihenfolge und Häufigkeit ausgeführt werden. Eine zielorientierte Modellierung bedeutet, dass für jeden unstrukturierten Prozess Ziele in Form von abzuliefernden Ausgabeobjekten in einem vorgegeben Qualitätsbereich bis zu einem bestimmten Zeitpunkt festgelegt werden. Man modelliert also nicht in prozessorientierter Weise die möglichen Bearbeitungswege, das „Wie“, sondern das Ergebnis oder Ziel, also das „Wann“ und „Wo“.

Ein Subprozessschritt innerhalb einer zielorientierten Aktivität wird *Schrittprogramm* genannt. Ein Schrittprogramm wird wiederum als zielorientierte Aktivität behandelt. Näheres wird in Abschnitt 6.2.3 beschrieben.

Ein Workflow-Modell ist für praktische Anwendungen nur dann geeignet, wenn Wiederverwendbarkeit und Strukturierbarkeit gewährleistet wird. Nur dann lassen sich aus bereits modellierten Aktivitäten und Workflows komplexere Workflows aufbauen. Zielorientierte Aktivitäten müssen deshalb analog zu Programmschritten in „klassischen“ Workflow-Management-Systemen zu einem strukturierten Gesamt-Workflow verknüpft werden können. Wie aus Abbildung 6.1 ersichtlich, müssen analog zu anderen prozessorientierten Ansätzen auch bei zielorientierten Aktivitäten die benötigten Ein- und Ausgabeobjekte, Schrittprogramme und Rollen festgelegt werden. Da sie unstrukturierte Teilprozesse repräsentieren und vorzeitige Datenweitergabe ermöglichen sollen, müssen jedoch zusätzliche Aspekte bei der Modellierung berücksichtigt werden, die in den folgenden Abschnitten beschrieben werden.

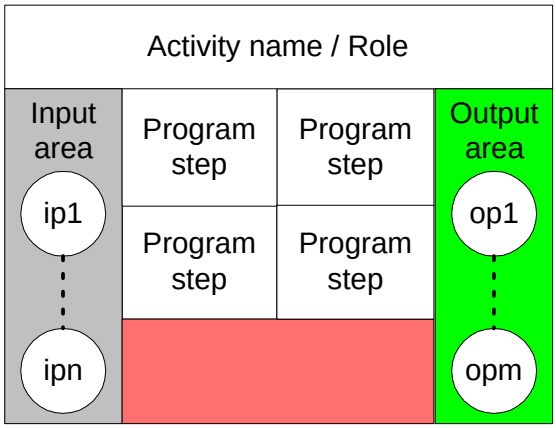



**Abbildung 6.1:** Modellierungsaspekte einer zielorientierten Aktivität

## 6.2 Zielorientierte Aktivitäten

Zielorientierte Aktivitäten repräsentieren unstrukturierte Teilprozesse und bilden die Basisbausteine zum Aufbau größerer Workflows im **WEP**-Modell. In der formalen Beschreibung besitzen zielorientierte Aktivitäten den folgenden Aufbau, der die soeben erläuterten Aspekte widerspiegelt:

$$
\begin{aligned}
Activity &\rightarrow Aspec \\
Aspec &\rightarrow (Aname, \\
&\quad Adescription, \\
&\quad APinput, \\
&\quad APoutput, \\
&\quad Amilestone, \\
&\quad Areturncode, \\
&\quad AprogramStep, \\
&\quad Arole)
\end{aligned}
$$

*Activity:*
: Eine Aktivität wird als Tupel – dargestellt durch die Terminale ( und ) – beschrieben.

$Aname \in UNIQUE$*:*
: ist der eindeutige Name der Aktivität. Die Menge $UNIQUE$ bezeichnet eine Menge aus eindeutigen Bezeichnern.

*Adescription:*
: ermöglicht eine Aufgabenbeschreibung der Aktivität in Textform.

*APinput, APoutput, Amilestone, Areturncode, AprogramStep, Arole:*
: Diese Elemente repräsentieren die Eingabe- beziehungsweise Ausgabeparameter, Aktivitätenmeilensteine und -Returncodes, Subprozessschritte sowie Rollenfüller. Sie werden in den folgenden Abschnitten beschrieben.

Basierend auf diesen Regeln lässt sich die Menge aller Aktivitäten eines beliebigen **WEP**-Workflows $WF \in$ **WEP**-$WORKFLOWS$ als Teilmenge der Menge aller erlaubten Wörter der **WEP**-Sprache $\mathbf{T}^*_{\mathbf{WEP}}$ definieren:

$$ACTIVITIES(WF) := \{w \in \mathbf{T}^*_{\mathbf{WEP}} : Activity \rightarrow^* w \ \wedge w \ IsPartialWordOf \ WF\}^1$$

Die Funktion $x \ IsPartialWordOf \ y$ liefert ***true***, wenn das Wort $x$ im Wort $y$ enthalten ist. Aufgrund der Vorgabe, dass jedes Element eines **WEP**-Workflows einen eindeutigen Namen $\in UNIQUE$ besitzt, ist diese Funktion immer injektiv. Sie findet also maximal nur einen „Treffer“ im Wort $y$.

Mit $ACTIVITIES$ wird dann die Menge aller Aktivitäten aus allen **WEP**-Workflows bezeichnet:

$$ACTIVITIES := \bigcup ACTIVITIES(WF) \ \forall WF \in \textbf{WEP-}WORKFLOWS$$

### 6.2.1 Modellierung von Eingabeparametern

Eine wichtige Grundlage für die Wiederverwendung der Aktivitäten als Basisbausteine ist eine formale Beschreibung der Ein- und Ausgabeparameter einer zielorientierten Aktivität. Auf der formalen Beschreibung der Parameter basiert später die Definition und Überprüfung korrekter Datenflüsse (siehe Abschnitt 8). Neben „Standardinformationen“ wie *Parametername* und *Objektklasse* erfordert eine vorzeitige Datenweitergabe zur Laufzeit zusätzlichen Modellierungsaufwand, der nun detaillierter beschrieben wird.

Die folgenden Regeln spezifizieren die Eingabeparameter von Aktivitäten als Menge, wobei jeder einzelne Eingabeparameter formal als ein Tupel dargestellt wird:

$$\begin{aligned}
APinput &\rightarrow \{APIset\} \\
APIset &\rightarrow APIspec \mid APIspec, APIset \\
APIspec &\rightarrow (APIname \\
&\quad\ APIclass, \\
&\quad\ APIqRange, \\
&\quad\ APIimportance, \\
&\quad\ APIconsistencyPolicy) \\
APIqRange &\rightarrow [OqLevel, OqLevel]
\end{aligned}$$

Die Terminale { und } dienen zur Repräsentation von Mengen. Die anderen Terminale müssen die folgenden Bedingungen erfüllen:

$APIname \in UNIQUE$:
ist der logische eindeutige Name des Parameters.

$APIclass \in OCLASS$:
Objekttyp des Parameters der Aktivität.

[1] Mit $\rightarrow^*$ wird die reflexive und transitive Hülle von $\rightarrow$ bezeichnet [HMU00].

$APIqRange$:
: Da beim **WEP**-Ansatz zielorientierte Aktivitäten bereits mit vorläufigen Eingabedaten bearbeitet werden können, muss die minimal und maximal benötigte Datenqualität spezifiziert werden, die zum Aufruf der Aktivität mindestens bzw. höchstens vorliegen muss. Dieser *Datenqualitätsbereich* (engl. *Quality Range*) wird durch Angabe zweier Qualitätsstufen in $[min : max]$-Notation ([$OqLevel$, $OqLevel$ ]) beschrieben. Die Informationen werden nicht nur zur *Laufzeit* verwendet, sondern werden schon während der *Modellierungszeit* bei Plausibilitätsprüfungen des Datenflusses mit berücksichtigt. Aus diesem Grund erfolgt die Bereichsspezifikation nur auf bereits zur Modellierungszeit auswertbaren Informationen (Namen von Qualitätsstufen, siehe Kapitel 8). Bei Ansätzen (z. B. [RMHN94]), die Datenqualitätsbereiche zusätzlich über erst zur Laufzeit auswertbare Funktionen spezifizieren, sind solche Plausibilitätsprüfungen dagegen nicht möglich.

$APIimportance \in APIMP$:
: Die Menge $APIMP$ (*Importance*) besteht aus den zwei Elementen $apimpRequired$ und $apimpOptional$. Ein Aufruf der Aktivität kann nur erfolgen, wenn alle als $apimpRequired$ definierte Eingabe-Objekte mit einer Qualitätsstufe innerhalb des Qualitätsbereichs vorhanden sind. Die Vorgabe ist $apimpRequired$. Die Unterscheidung zwischen optionalen und zwingend erforderlichen Eingabeparametern eröffnet zusätzliche Modellierungsmöglichkeiten. So können bei Aktivitäten Eingabeparameter modelliert werden, die nicht notwendigerweise für ihre (erste) Bearbeitung vorhanden sein müssen, deren (späteres) Eintreffen aber die Bearbeitung erleichtert. Auch können Workflows modelliert werden, bei denen eine Aktivität die optionalen Eingabeparameter nicht auf allen Wegen oder nur beim wiederholten Aktivieren (über Schleifen) erhält.

$APIconsistencyPolicy$:
: Hiermit wird spezifiziert, nach welcher Strategie ($acpMerge$ oder $acpUndoRedo$) neu eingetroffene Eingabeobjekte besserer Datenqualität in eine bereits laufende Aktivität integriert werden können. Dies wird detaillierter in Kapitel 12 beschrieben.

Basierend auf diesen Regeln lässt sich die Menge der Eingabeparameter einer Aktivität $A$ $APINPUTS(A)$ als Teilmenge der Menge aller erlaubten Wörter der **WEP**-Sprache $\mathbf{T}^*_{\mathbf{WEP}}$ definieren.

$$\begin{array}{l} \text{Sei } A \in Activity \\ APINPUTS(A) \;:=\; \{w \in \mathbf{T}^*_{\mathbf{WEP}} : APinput \rightarrow^* w \;\wedge w\, IsPartialWordOf\; A\} \end{array}$$

### 6.2.2 Modellierung von Ausgabeparametern, Meilensteinen und Returncodes

Die Modellierung der abzuliefernden Ergebnisse einer zielorientierten Aktivität gestaltet sich aufgrund der Möglichkeiten vorzeitiger Datenweitergabe schwieriger als bei herkömmlichen Workflow-Modellen. Sie gliedert sich in drei Aspekte, der Modellierung von Ausgabeparametern, der Festlegung von Zwischenergebnissen in Form von Meilensteinen und der Beschreibung von Returncodes, die potenzielle Endergebnisse einer zielorientierten Aktivität repräsentieren.

#### 6.2.2.1 Modellierung von Ausgabeparametern

Wie in herkömmlichen Workflow-Modellen wird auch im **WEP**-Modell ein Ausgabeparameter durch Parametername und Objektklasse definiert. Formal wird die Menge aller Ausgabeparameter einer zielorientierten Aktivität durch folgende Regeln spezifiziert:

$$\begin{aligned} APoutput &\rightarrow \{APOset\} \\ APOset &\rightarrow APOspec \mid APOspec, APOset \\ APOspec &\rightarrow (APOname, APOclass) \end{aligned}$$

Basierend auf diesen Regeln lässt sich analog zu den Eingabeparametern die Menge der Ausgabeparameter einer Aktivität $A$ $APOUTPUTS(A)$ definieren.

$$\begin{aligned} &\text{Sei } A \in Activity \\ APOUTPUTS(A) &:= \{w \in \mathbf{T}^*_{\mathbf{WEP}} : APoutput \rightarrow^* w \ \wedge w \, IsPartialWordOf \ A\} \end{aligned}$$

#### 6.2.2.2 Modellierung von Meilensteinen

Für ein kontrolliertes Weitergeben vorläufiger Ausgabeobjekte sind weitere Aspekte zu berücksichtigen:

Erstens muss definiert werden, was ein „sinnvoller Bestand“ für die Weitergabe darstellt. Darunter versteht man, welche Ausgabeparameter in welchen Datenqualitätsbereichen bereitgestellt werden müssen.

Zweitens muss festgelegt werden, wann die Weitergabe der Daten zu erfolgen hat. Hierbei sind nur relative Zeitangaben zu einem festzulegenden Bezugspunkt sinnvoll. Als Bezugspunkte können das Anbieten einer Aktivität in den Arbeitslisten der betroffenen Personen (*Anbietungszeitpunkt*) oder der *Startzeitpunkt* der Aktivitätenbearbeitung verwendet werden. Für die weitere Betrachtung des Modells ist die konkrete Wahl des Bezugspunkts ohne Belang, da jederzeit über eine Relativangabe auf den anderen Bezugspunkt referenziert werden kann. Da es in der Praxis häufig schwierig ist, die Zeit für die Wartezeit einer Aktivität zu bestimmen, werden diese meist gar nicht erfasst.[2] Die Bestimmung der reinen Bearbeitungszeit ist in der Regel einfacher. Aus diesem Grunde beziehen sich beim **WEP**-Modell alle Relativangaben auf den Startzeitpunkt einer Aktivität.

Darüber hinaus muss die Zusammenarbeit während der durch vorläufige Datenweitergabe möglichen Simultaneous-Engineering-Phasen geregelt werden. In Abschnitt 6.2.1 wurde festlegt, wie ein Bearbeiter mit dem Eintreffen neuerer Datenobjekte umzugehen hat. Es muss nun noch definiert werden, wie bei der wiederholten Weitergabe von vorläufigen Ausgabeobjekte an Folgeschritte zu verfahren ist. Denkbar sind hier eine völlig autonome Weitergabe der Daten oder eine vorhergehende Beratung mit den betroffenen Bearbeitern nachfolgender Prozessschritte, bevor die aktuelleren Datenobjekte weitergegeben wurden, so dass diese Änderungswünsche bezüglich des Inhalts einbringen können. Da hier die sinnvolle Strategie sehr von der Aufgabe des Prozessschritts beziehungsweise von dem

[2] beziehungsweise wird nicht zwischen Warte- und Bearbeitungszeit unterschieden und die Bearbeitungszeit um die Wartezeit erhöht

zu erwartenden Integrationsaufwand bei den nachfolgenden Schritten abhängt, muss für einen praktikablen Einsatz die Weitergabestrategie für jede Aktivität konfigurierbar sein. Neben einer völlig autonomen Weitergabe der Objekte benötigt man sicherlich auch eine Strategie, die mehr Mitbestimmungsrecht für die nachfolgenden Schritte ermöglicht. Hierfür sind Strategien aus dem Gebiet der Replikaktionsverfahren (siehe [BD95, BD96b] für eine Übersicht) nach Adaption auf die spezifischen Anforderungen anwendbar. Die Weitergabestrategie im **WEP**-Modell wird durch das Merkmal *ConcurrentMode* spezifiziert (siehe unten). Eine detaillierte Beschreibung des Laufzeitverhaltens ist in Kapitel 12 zu finden.

Es liegt in der Natur der Sache, dass die Weitergabe vorläufiger Informationen auch Gefahren birgt, da vorläufige Daten zum Teil falsche oder zumindest unvollständige Informationen enthalten, deren Weitergabe auf einige wenige Folgeprozessschritte eingeschränkt werden sollte. Auch dieser Aspekt wird im **WEP**-Modell berücksichtigt und lässt sich über das Merkmal *Distance* konfigurieren.

Zur Modellierung aller dieser Aspekte werden im **WEP**-Modell *Meilensteine* eingeführt. Ein Meilenstein einer Aktivität besitzt einen Namen, eine Menge von *qObjekten*[3] und einen Bearbeitungszeitraum. Die formale Beschreibung genügt den folgenden Regeln:

$$
\begin{aligned}
Amilestone &\rightarrow \{AMset\} \\
AMset &\rightarrow AMspec | AMspec, AMset \\
AMspec &\rightarrow (AMname, \\
&\quad \{AMobjSpec\}, \\
&\quad AMtimeSpec) \\
AMobjSpec &\rightarrow AMqObjSpec \mid \\
&\quad AMqObjSpec, AMobjSpec \qquad \circledast \\
AMqObjSpec &\rightarrow (APOspec, AMqLevel, \\
&\quad AMconcurrentMode, AMdistance) \\
AMqLevel &\rightarrow OqLevel
\end{aligned}
$$

*$Amilestone$:*
Einer Aktivität kann eine Menge von Meilensteinen ($\{AMset\}$) zugeordnet werden. Jeder Meilenstein einer Aktivität im **WEP**-Modell besteht aus den Komponenten $AMname$, einer Menge zugeordneter qObjekte $AMobjSpec$ und einer Zeitangabe $AMtimeSpec$.

*$AMname$:*
$AMname \in UNIQUE$ bezeichnet den eindeutigen Namen des Meilensteins.

*$AMobjSpec$:*
Die mit $\circledast$ markierte Regel ermöglicht die Auflistung mehrerer qObjekt-Tupel $AMqObjSpec$ für einen Meilenstein.

*$AMqObjSpec$:*
Hiermit werden qObjekt-Tupel spezifiziert, die aus folgenden Elementen bestehen:

*$APOspec$:*
$APOspec$ ist ein beliebiger Ausgabeparameter aus der Menge aller Ausgabeparameter. Für jeden Meilenstein einer Aktivität $A$ muss gelten: $APOspec \in APOUTPUTS(A)$.

[3] Mit *qObjekt* bezeichnet man ein Objekt, wenn eine Qualitätsstufe bzw. ein Qualitätsbereich angegeben ist.

$AMqLevel$:

$AMqLevel$ spezifiziert die geforderte Qualitätsstufe $OqLevel$ des Ausgabeparameters für diesen Meilenstein.

$AMconcurrentMode$:

$AMconcurrentMode \in \{AMcmAutonomous, AMcmConsolidation\}$. Dieses Attribut beschreibt die Vorgehensweise bei der Behandlung vorzeitig weitergegebener Daten. Hat es den Wert $AMcmAutonomous$, wird das Ausgabeobjekt eigenmächtig weitergegeben. Abhängige Folgeaktivitäten werden erst nach der Änderung informiert. Im Modus $AMcmConsolidation$ wird eine sogenannte Konsolidierungsrunde (siehe Kapitel 12) anberaumt, während der alle Beteiligten über das weiterzugebende Objekt beraten können. Die letztendliche Entscheidungsgewalt liegt jedoch auch hier beim Bearbeiter dieser Aktivität.

Die Belegung des Merkmals $AMconcurrentMode$ mit dem Wert $AMcmConsolidation$ macht immer dann Sinn, wenn durch die Aktualisierung einer bereits weitergegebenen Objektversion bei den lesenden Aktivitäten eventuell erheblicher zusätzlicher Arbeitsaufwand verursacht würde. Eine *vorherige* Rücksprache mit den betroffenen Bearbeitern ist hier wichtig, in der sie ihren Aufwand darlegen können, um damit einen für alle tragfähigen Kompromiss finden zu können.

Mit Hilfe dieses Merkmals sind Workflow-Modellierer in der Lage situationsbedingte Ad-hoc-Abstimmungsprozesse, wie sie in Teil I im Rahmen einer Simultaneous-Engineering-Unterstützung von Produktentwicklungsprozessen gefordert wurden, durch das **WEP**-Workflow-Management-System automatisch anzustoßen.

$AMdistance$:

Das Merkmal $AMdistance \in \mathbb{N}$ legt die Entfernung, wie weit sich vorläufige Daten von der aktuellen Aktivität entfernen dürfen, *in Anzahl Aktivitäten* fest.

$AMtimeSpec \in \mathbb{N}$:

Mittels der Zeitspanne modelliert man den *durchschnittlichen* Bearbeitungszeitraum, der zur Verfügung steht, um die zum Meilenstein spezifizierten Ausgabeobjekte in der jeweiligen Qualitätsstufe bereitzustellen.

Wiederum bezeichnet $AMILESTONES(A)$ die Menge aller Meilensteine einer Aktivität $A$.

Die Meilensteine einer Aktivität sind bezüglich der Zeitvorgaben geordnet. Bei der Modellierung eines Meilensteins muss deshalb – will man das in Teil I geforderte Prinzip der inkrementellen Konkretisierung umsetzen – sichergestellt werden, dass die spezifizierten Ausgabeobjekte mindestens die gleich hohe Qualität besitzen wie dieselben Ausgabeobjekte zeitlich früherer Meilensteine.

*Definition 1: Modellierungsregel „inkrementelle Konkretisierung“*

Seien $ms_1, ms_2 \in AMILESTONES(A)$ beliebige Meilensteine der Aktivität $A$,

$$msTime_i := w \in \mathbf{T}^*_{\mathbf{WEP}} : AMtimeSpec \rightarrow^* w \ \wedge \ w\, IsPartialWordOf\ ms_i,\ i = 1, 2$$

die Bearbeitungszeiträume von $ms_1$ und $ms_2$. (o. B. d. A. sei $msTime_1 \leq msTime_2$) sowie

$$amQobjSpec(ms_i) := \{w \in \mathbf{T}^*_{\mathbf{WEP}} : AMqObjSpec \rightarrow^* w \wedge w\, IsPartialWordOf\ ms_i\},\ i = 1, 2$$

die Menge aller qObjekte von $ms_1$ beziehungsweise $ms_2$.

Für jedes qObjekt, dessen Ausgabeparameterspezifikation ($APOspec$) in beiden Meilensteinen enthalten ist, für das also gilt:

$$\begin{aligned}\exists\, outputObj: \quad & APOspec \rightarrow^* outputObj \wedge \\ & outputObj\ IsPartialWordOf\ amQobjSpec(ms_1) \wedge \\ & outputObj\ IsPartialWordOf\ amQobjSpec(ms_2),\end{aligned}$$

muss gelten:

$$outputObjQlevel_1 <_{QL} outputObjQlevel_2,$$

wobei gilt:

$$\begin{aligned}outputObjQlevel_1 \quad := \quad & AMqLevel \rightarrow^* outputObjQlevel_1 \ \wedge \\ & outputObjQlevel_1\ IsPartialWordOf\ amQobjSpec(ms_1) \\ outputObjQlevel_2 \quad := \quad & AMqLevel \rightarrow^* outputObjQlevel_2 \ \wedge \\ & outputObjQlevel_2\ IsPartialWordOf\ amQobjSpec(ms_2)\end{aligned}$$

Das folgende Beispiel illustriert den Sachverhalt:

*Beispiel 1: nach Definition 1 konforme Meilensteine*

$$\begin{aligned}\{(ms_1,\{&((op_1, oc_1), oql_5, AMcmAutonomous, 3),\\ &((op_7, oc_7), oql_3, AMcmAutonomous, 5)\}, 10)\\ (ms_2,\{&((op_3, oc_3), oql_4, AMcmConsolidation, 1),\\ &((op_7, oc_7), oql_8, AMcmAutonomous, 5)\}, 15)\}\end{aligned}$$

Der Meilenstein $ms_1$ ist als erster zu erfüllen. Das Ausgabeobjekt, das in beiden Meilensteinen referenziert wird, ist $op_7$. Die Qualitätsstufe von $op_7$ ist im Meilenstein $ms_1$ kleiner als die Qualitätsstufe von $op_7$ im Meilenstein $ms_2$.

#### 6.2.2.3 Modellierung von Returncodes

Bei der Festlegung von Returncodes einer Aktivität sind die folgenden Anforderungen zu erfüllen:

1. Eine Aktivität kann mit unterschiedlichen Ergebnissen beendet werden. Dadurch lassen sich Workflows mit bedingten Verzweigungen und Schleifen modellieren.
2. Jedem Ergebnis müssen verschiedene Ausgabeobjekte mit verschiedenen Qualitätsbereichen zugeordnet werden können.
3. Jedes Ergebnis einer Aktivität soll – analog zu den Zwischenergebnissen der Meilensteine – ebenfalls zeitlich überwacht werden.

Anforderung 1 wird üblicherweise durch verschiedene Returncodes spezifiziert. Dagegen ist Anforderung 2 in der Regel in herkömmlichen Workflow-Management-Systemen nicht realisierbar. Hier wird meist von der Annahme ausgegangen, dass alle als notwendig spezifizierten Ausgabeobjekte geeignet belegt sind, unabhängig davon, mit welchem Returncode die Aktivität beendet wurde. Aus diesem Grund muss in der Praxis häufig mit „Dummy“-Parameterbelegungen gearbeitet werden, deren Semantik den nachfolgenden Aktivitäten bekannt sein muss. Diese *implizite* Umsetzung der Anforderung 2 ist insbesondere bei langdauernden Aktivitäten, bei denen die abzuliefernden Ergebnisobjekte

stark vom Ausgang (und damit vom Returncode) abhängen, inadäquat sowie wenig transparent und somit durch das Workflow-Management-System nicht überprüfbar.

Deshalb wird im **WEP**-Modell ein *expliziter* Ansatz über die bereits bekannten Meilensteine gewählt. Damit wird für jeden Returncode einer Aktivität spezifiziert, welche Ausgabeobjekte in welchen Qualitätsbereichen bereitgestellt werden sollen. Durch diese *explizite* und damit dem Workflow-Management-System bekannte Modellierung kann die vom jeweiligen Returncode abhängige Vollständigkeit der bereitgestellten Datenobjekte auch durch das Workflow-Management-System überwacht werden. Zusätzlich lässt sich einerseits Bearbeitungszeit einsparen, beispielsweise bei geringeren Datenanforderungen für ein Bearbeitungsergebnis, und anderseits sicherstellen, dass eine Aktivität nicht ohne die erforderlichen Daten beendet werden kann.

Durch diesen Ansatz über Meilensteine konnte gleichzeitig die Anforderung 3 einer zeitlichen Überwachung der Ergebnisbereitstellung umgesetzt werden.

In der formalen Spezifikation wird die Menge der Returncodes von Aktivitäten beschrieben durch:

$$\begin{aligned} Areturncode &\rightarrow \{ARCset\} \\ ARCset &\rightarrow ARCspec \mid ARCspec, ARCset \\ ARCspec &\rightarrow (ARCname, \{ARCmsSet\}) \\ ARCmsSet &\rightarrow AMspec \mid AMspec, ARCmsSet \end{aligned}$$

*$ARCname$:*
: $ARCname \in UNIQUE$ bezeichnet den eindeutigen Namen des Returncodes.

*$ARCmsSet$:*
: $ARCmsSet$ repräsentiert eine beliebige Teilmenge von Meilensteinen aus der Menge aller Meilensteine. Für jeden Meilenstein einer Aktivität $A$ muss gelten: $AMspec \in AMILESTONES(A)$. Der Meilenstein mit dem höchsten Zeitlimit stellt das Endergebnis der Aktivität für diesen Returncode dar. Die anderen Meilensteine bezeichnen dagegen *zwingend* erforderliche Zwischenergebnisse. Die Notwendigkeit für diese zwingend erforderlichen Zwischenergebnisse wird bei der Einführung der Datenfluss-Spezifikation des **WEP**-Modells erkennbar.

Mit $RETURNCODES(A) := \{w \in \mathbf{T}^*_{\mathbf{WEP}} : Areturncode \rightarrow^* w \wedge w\, IsPartialWordOf\, A\}$ wird die Menge aller Returncodes einer Aktivität $A$ bezeichnet.

Benötigt man zum Erreichen unterschiedlicher Ergebnisse einer Aktivität verschieden lange Bearbeitungszeiten, ordnet man also den einzelnen Returncodes Meilensteine mit verschiedenen Zeitlimits zu, so führt dies während der Laufzeit zur Verletzung von Zeitrestriktionen, wenn nicht der Returncode mit der kürzesten Bearbeitungszeit gewählt wird. Diese Problematik soll an dieser Stelle anhand des folgenden Beispiels aufgezeigt werden.

*Beispiel 2: Returncodes mit unterschiedlichen Zeitvorgaben*

Es seien die in Abbildung 6.2 gezeigten Aktivitäten gegeben, die über eine bedingte Verzweigung miteinander verbunden sind. Wird bei der Aktivität $A1$ der Returncode $rc_1$ gewählt, so wird Aktivität $A2$ aktiviert, andernfalls die Aktivität $A3$.

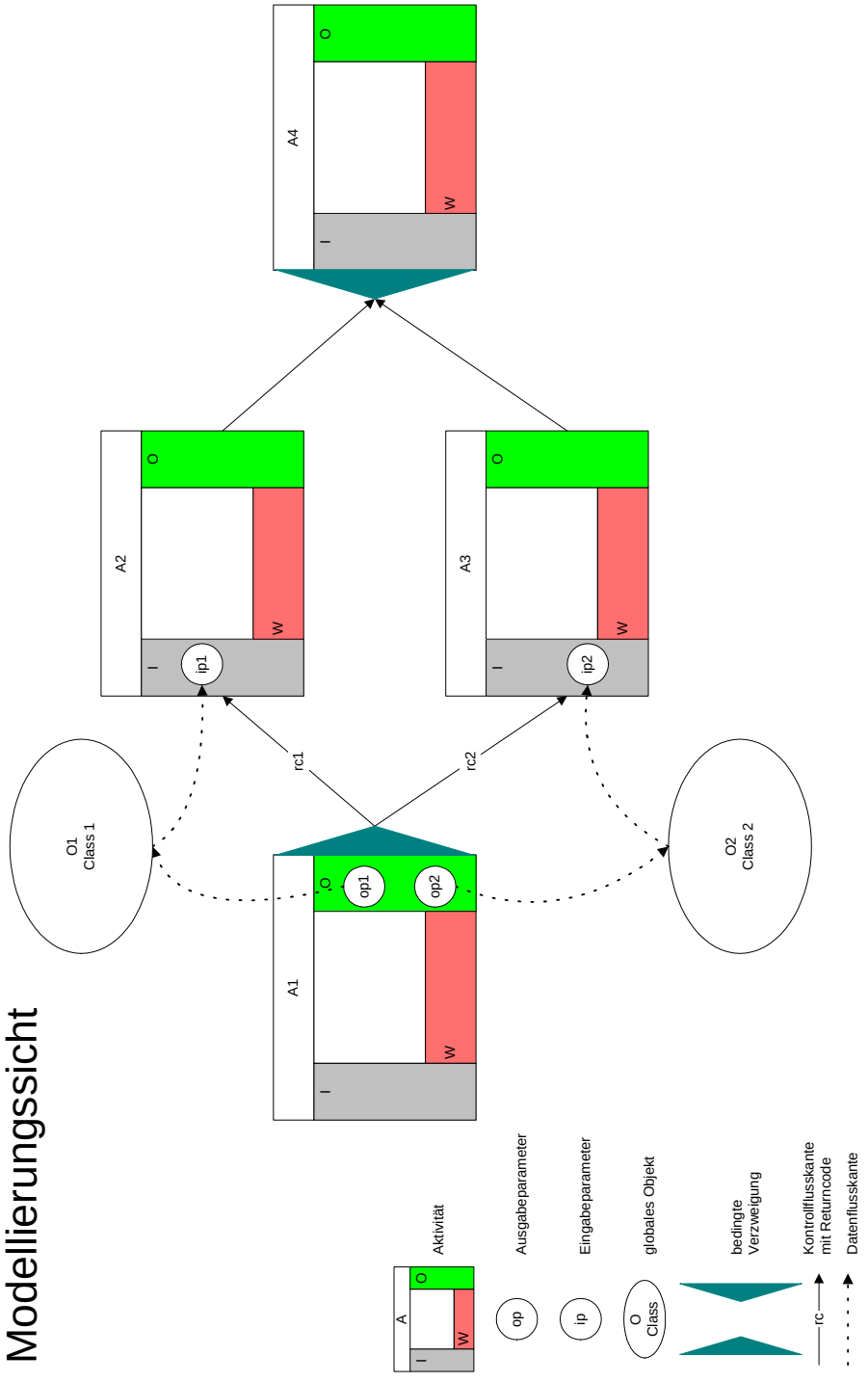



**Abbildung 6.2:** Beispiel von Returncodes mit unterschiedlichen Bearbeitungsdauern und Ausgabeobjekten

Bei der Aktivität $A1$ sind drei Meilensteine und zwei Returncodes definiert:

$$\{(ms1, \{((op1, oc1), oQl5, \ldots, \ldots), ((op2, oc2), oQl3, \ldots, \ldots)\}, 10),$$
$$(ms2, \{((op1, oc1), oQl6, \ldots, \ldots)\}, 20),$$
$$(ms3, \{((op2, oc2), oQl8, \ldots, \ldots)\}, 30)\}$$
$$\{(rc1, \{(ms2, \ldots)\}), (rc2, \{(ms3, \ldots)\})\}$$

Die Aktivitäten $A2$ und $A3$ benötigen die Eingabeparameter der Klassen $oc1$ beziehungsweise $oc2$:

$$(A2, \ldots, \{(ip1, oc1), [oql1, oql8], apimpRequired, \ldots)\}, \ldots, \ldots)$$
$$(A3, \ldots, \{(ip2, oc2), [oql1, oql7], apimpRequired, \ldots)\}, \ldots, \ldots)$$

Mit Ablauf des Zeitlimits von Meilenstein $ms1$ werden die Aktivitäten $A2$ und $A3$ mit vorläufigen Datenobjekten versorgt: $A2$ mit einer Objektversion aus Objekt $O1$ in der Qualitätsstufe $oql5$ und $A3$ mit einer Objektversion aus Objekt $O2$ in der Qualitätsstufe $oql3$.

Entscheidet sich der Bearbeiter von Aktivität $A1$, dass er die Aktivität später mit Returncode $rc2$ beenden möchte, bei dem nur das Objekt $O2$ relevant ist, so wird er keine Objektversionen für das Objekt $O1$ in der Qualitätsstufe $oql6$ bereitstellen. Damit wird der Meilenstein $ms2$ verletzt werden und die Aktivität $A2$ erhält keine verbesserte Objektversion. Dies ist jedoch nicht von Belang, wenn später tatsächlich Returncode $rc2$ gewählt wird. Denn dann wird die Arbeit der Aktivität $A2$ überflüssig und sie wird zurückgesetzt. Durch die Nicht-Bereitstellung einer neuen Objektversion für $A2$ erspart man sich unnötige Arbeit.

Entscheidet sich der Aktivitätenbearbeiter – entgegen seiner ursprünglichen Absicht – später doch für den mit Returncode $rc1$ spezifizierten Pfad (Pfad zu $A2$), so müssen die Objekte des zugeordneten Meilensteins $ms2$, also $O1$ in der Qualitätsstufe $oql6$, nachgereicht werden, wodurch die Prozessdurchlaufzeit verlängert wird, weil $A2$ die benötigten Daten später erhält.

Die Verknüpfung von Meilensteinen zu Returncodes führt also zu *„bedingten“ Meilensteinen*. Dies sind Meilensteine, die nur für bestimmte Returncodes relevant sind (vergleiche Abbildung 6.3). Die Modellierungsregel 1 wird dann für jeden Returncode-Zweig einzeln angewandt.

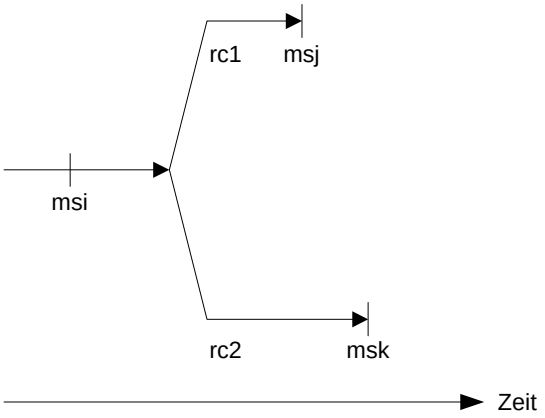



**Abbildung 6.3:** Zusammenhang Meilensteine und Returncodes: Die Modellierungsregel 1 muss nur für die Meilensteine $msi, msj$ für Returncode $rc1$ und $msi, msk$ für Returncode $rc2$ gültig sein.

### 6.2.3 Definition von Schrittprogrammen

Jedes Schrittprogramm (*program step*) repräsentiert einen Subprozessschritt innerhalb eines unstrukturierten Teilprozesses (vergleiche Abschnitt 6.1). Die Menge der Schrittprogramme bildet die Basis zur Modellierung von unstrukturierten Teilprozessen. Da sich die Reihenfolge und Häufigkeit der Aktivierung der Schrittprogramme erst zur Laufzeit ergeben, wird zwischen Schrittprogrammen weder Kontroll- noch Datenfluss modelliert. Die Modellierung von Schrittprogrammen beschränkt sich im Wesentlichen auf die Festlegung von Ein- und Ausgabeparametern sowie deren Objektklassen.

Ein Schrittprogramm kann repräsentieren:

- ein *Anwendungssystem* (z.B. Texteditor, CAD-System). Das Anwendungssystem ist eventuell von einem *Wrapper* umgegeben, um Parameterversorgung und -rückgabe zu gewährleisten. Bei der Spezifikation muss eine Aufrufbeschreibung (Pfad, Programmname, ...) angegeben werden.
- eine zielorientierte Aktivität beziehungsweise ein kompletter **WEP**-Workflow. Hier genügt als weitere Angabe die Referenz auf eine bereits modellierte zielorientierte Aktivität beziehungsweise auf einen **WEP**-Workflow.

Auf die Belegung der formalen Parameter mit aktuellen Werten wird im Teil III eingegangen.

Die formale Spezifikation ist wie folgt:

$$
\begin{aligned}
AprogramStep &\rightarrow \{APSset\} \\
APSset &\rightarrow APSspec | APSspec, APSset \\
APSspec &\rightarrow APStool | Activity | Workflow \\
APStool &\rightarrow (APStoolName, APinput, APoutput, \ldots)
\end{aligned}
$$

Die einzelnen Symbole bedeuten:

$APStool$:
: $APStool$ referenziert auf ein Anwendungssystem, das mittels $APStoolName \in UNIQUE$, der Spezifikation der Eingabe- und Ausgabeparameter – analog zu zielorientierten Aktivitäten – und weiteren Informationen, die für das weitere Verständnis hier irrelevant sind, beschrieben wird.

$Activity \in ACTIVITIES$:
: $Activity$ referenziert auf eine bereits modellierte zielorientierte Aktivität.

$Workflow \in$ ***WEP***-$WORKFLOWS$:
: $Workflow$ referenziert auf einen bereits modellierten **WEP**-Workflow.

Die Spezifikation einer zielorientierten Aktivität beziehungsweise eines **WEP**-Workflows für ein Schrittprogramm erweitert die Einsatzmöglichkeiten des **WEP**-Modells enorm (vergleiche Abbildung 6.4):

Einerseits lassen sich völlig *unstrukturierte* Prozesse („Ad-hoc-Prozesse“) gestalten, in dem der **WEP**-Workflow aus nur einer zielorientierten Aktivität besteht, die jedoch alle im unstrukturierten Prozess erlaubten Prozessschritte als zielorientierte Aktivitäten beinhaltet.

Anderseits kann auch ein starr strukturierter Workflow modelliert werden. Hier besitzt jede zielorientierte Aktivität nur ein Schrittprogramm, das auf ein Anwendungsprogramm referenziert.

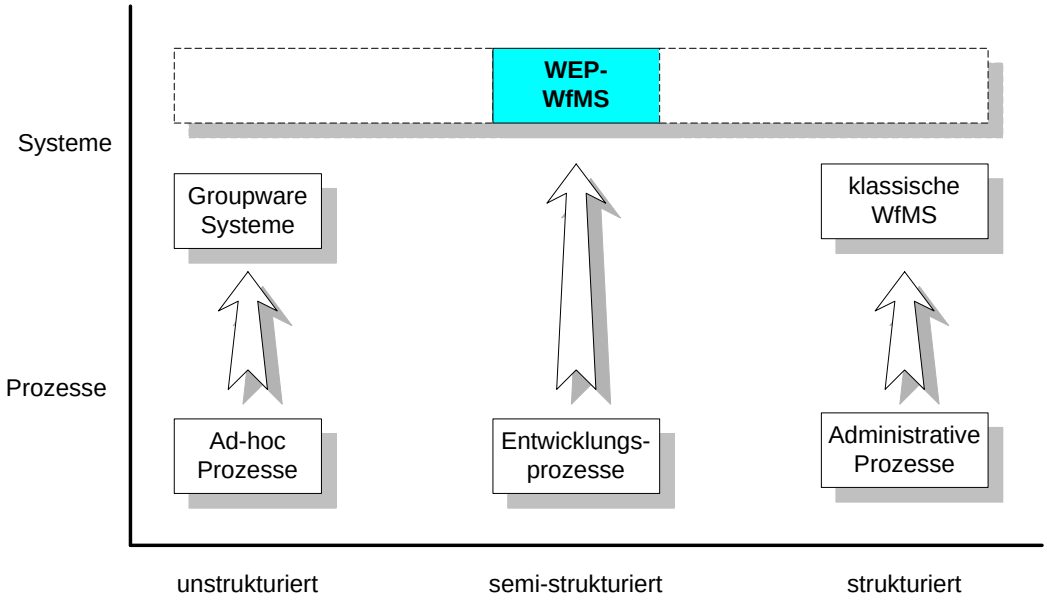



**Abbildung 6.4:** Erweiterungsmöglichkeiten des **WEP**-Modells auf andere Anwendungsfelder

### 6.2.4 Weitere Aspekte bei der Modellierung zielorientierter Aktivitäten

Natürlich sind noch weitere Aspekte für eine vollständige Modellierung von zielorientierten Aktivitäten zu behandeln. Neben einer informellen Beschreibung der Aufgabe für den Aktivitätenbearbeiter und einer sinnvollen Kategorisierung für eine Verwaltung in Aktivitäten-Repositories ist hier besonders die Spezifikation von Rollen zu nennen. Diese Aspekte bilden jedoch nicht Schwerpunkt der Arbeit. Sie sind für das weitere Verständnis auch nicht von Belang. Es sei hier auf die entsprechende Literatur verwiesen (z.B.: [Jab95, JB96, Ros96]). Für die Spezifikation von Rollen, Organisationseinheiten, Stellen, Kompetenzen ist beispielsweise in [Kub98] ein umfassender Vorschlag erarbeitet worden.

# Kapitel 7

# Kontrollflussmodellierung im WEP-Modell

Mittels der Modellierung des Kontrollflusses werden Ausführungsreihenfolgen und -bedingungen der einzelnen Workflow-Basisbausteine präzise festgelegt. Sie repräsentiert damit den verhaltensbezogenen Aspekt der Workflow-Modellierung [Jab95]. Im **WEP**-Modell bilden die zielorientierten Aktivitäten die Basisbausteine des Workflows. Ziel ist es, dass – wie in einem herkömmlichen Workflow-Management-System – die zielorientierten Aktivitäten des **WEP**-Modells zu einem Gesamtprozess verknüpft werden, um dadurch schrittweise komplexere Prozesse als **WEP**-Workflows umsetzen zu können.

Wichtig für die praktische Anwendbarkeit der Ablaufbeschreibungssprache sind sicherlich die Ausdrucksmächtigkeit und Verständlichkeit der angebotenen Modellierungskonstrukte für das anvisierte Anwendungsgebiet. Wie bereits im Teil I motiviert sind für Entwicklungsprozesse Konstrukte zur Modellierung *sequentieller*, *alternativer* sowie *statisch und variabel paralleler* Ausführungspfade notwendig. Die stark iterative Vorgehensweise bei der Produktentwicklung erfordert außerdem noch die Bereitstellung von Konstrukten zur Modellierung von *Schleifen*. Daneben ist aus Gründen der Orthogonalität und Komplexitätsreduzierung darauf zu achten, dass der Aspekt der Kontrollflussmodellierung möglichst getrennt von der Beschreibung anderer Aspekte, wie beispielsweise Daten- oder Zeitaspekte, erfolgen kann. Natürlich bestehen gewisse Abhängigkeiten zwischen verschiedenen Modellierungsaspekten. Man denke hierbei beispielsweise an die Datenversorgung von Aktivitätenparametern, die nicht gegen die Kontrollflussreihenfolge erfolgen kann. Diese Abhängigkeiten finden sich in Form von Modellierungsregeln im **WEP**-Modell wieder, die sukzessive im Laufe des Modellierungsteils eingeführt werden.

Aus Gründen der besseren Verständlichkeit werden dabei nicht alle Konstrukte des **WEP**-Modells gleich im Detail eingeführt. Modellierungskonstrukte, deren Notwendigkeit erst später nach der Behandlung weiterer Aspekte wie beispielsweise der Datenflussmodellierung oder der exakten Schaltlogik sichtbar wird, werden nur informell beschrieben und später präzisiert.

Im folgenden Abschnitt wird die grundlegende Philosophie des im **WEP**-Modell verfolgten Ansatzes zur strukturierten Modellierung des Kontrollflusses vorgestellt. Die darauffolgenden Abschnitte führen dann informell mittels strukturierter Graphen die bereitgestellten Kontrollflusskonstrukte ein, die dann im Abschnitt 7.3 formal durch eine Grammatik beschrieben werden. Dieser Formalismus

wird später benötigt, um mittels Analysen präzise und zuverlässige Aussagen über die Korrektheit der modellierten Workflows treffen zu können.

## 7.1 Strukturierte Modellierung von Workflows

Wie in anderen Workflow-Modellen auch [RBFD01, LA94], basiert die Kontrollflussmodellierung im **WEP**-Modell auf einem strukturierten, graphbasierten Ansatz. In Zentrum dieses Ansatzes steht das Konzept der regelmäßigen Blockstrukturierung [Dij68]. Ein **WEP**-Workflow wird hierbei aus Blöcken aufgebaut, die entweder *völlig ineinander geschachtelt* oder *disjunkt* sind. Eine andere Überlappung von Blöcken ist nicht möglich. Neben den zielorientierten Aktivitäten als „Elementarblöcke“ sind auch alle **WEP**-Kontrollflusskonstrukte Blöcke mit eindeutigem Start und Ende. Auf diese Weise ist ein so modellierter **WEP**-Workflow aus Kontrollflusssicht ebenfalls ein Block mit eindeutigem Start und Ende.

Blöcke bilden damit das fundamentale Konzept für die Modellierung strukturierter Workflows. Die Vorteile sind:

- Einfachere und übersichtlichere Modellierung durch bessere Benutzerführung basierend auf syntaxgesteuerten Workflow-Modellierungs-Editoren.
- Vermeidung beziehungsweise frühzeitige Erkennung von Modellierungsfehlern: In zahlreichen Fällen kann die Korrektheit und Konsistenz der Workflow-Modelle bereits „per Konstruktion“ sichergestellt werden, so dass aufwändige Analysen entfallen.
- Reduzierung der Komplexität durch einfache Wiederverwendung bereits vorhandener Kontrollflussblöcke beziehungsweise **WEP**-Workflow-Modelle (*Divide-and-conquer-Methode*)
- Effiziente Analyse und Verifikation eines modellierten Workflows. Jeder Block kann für sich überprüft werden. Ein bereits verifizierter Block kann als „Blackbox“ betrachtet werden, so dass bei der Überprüfung des übergeordneten Blocks nur noch die *Schnittstelle* und *nicht* mehr die *interne Struktur* des inneren Blocks analysiert werden muss.

Mittels dieses blockorientierten Ansatzes lassen sich mit den im **WEP**-Modell vorhandenen Kontrollflusskonstrukten in der Praxis die meisten Arbeitsabläufe adäquat abbilden. Es sind in der Theorie jedoch Konstellationen denkbar, die sich nicht auf ein blockorientiertes Prozessmodell abbilden lassen (vergleiche Abbildung 7.1). Solche Prozesse sind dem Autor aus dem Gebiet der Produktentwicklung jedoch nicht bekannt.

## 7.2 WEP-Konstrukte für die Kontrollflussmodellierung

Es werden nun die einzelnen Kontrollflusskonstrukte des **WEP**-Modells beschrieben.

### 7.2.1 Modellierung von Sequenzen

Die Sequenz (siehe Abbildung 7.2) beschreibt die Nacheinanderausführung von Aktivitäten im **WEP**-Modell. Jede Aktivität besitzt genau einen Vorgänger und einen Nachfolger. Sie sind durch gerichtete Kontrollflusskanten verbunden.

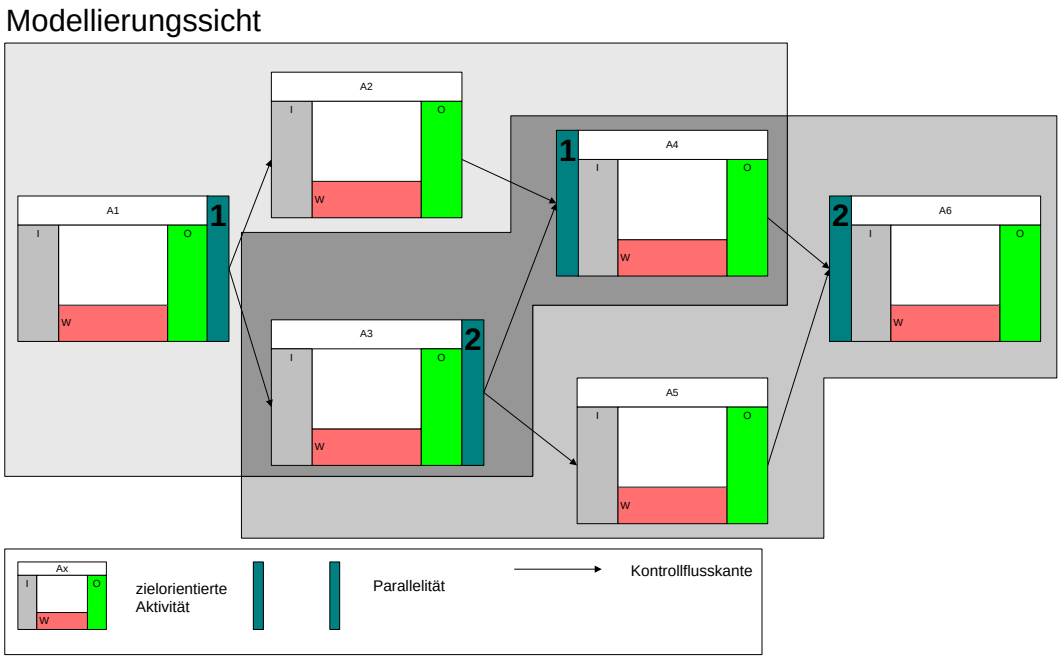



**Abbildung 7.1:** Beispiel für einen Prozess ohne vollständige Blockstruktur

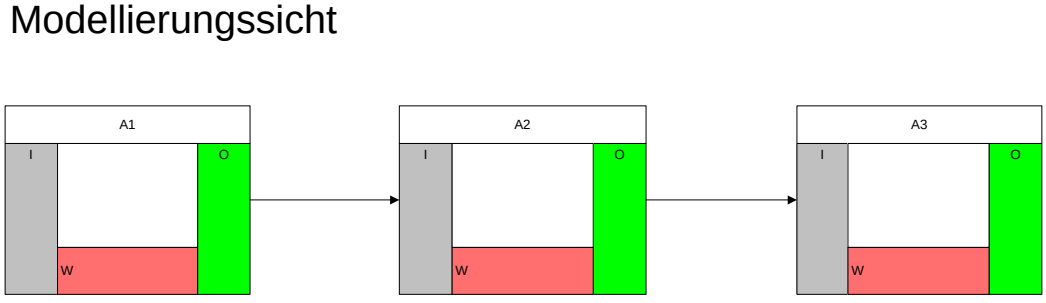



**Abbildung 7.2:** Sequenzkonstrukt im **WEP**-Modell

### 7.2.2 Modellierung bedingter Verzweigungen

Ein weiteres Kontrollflusskonstrukt im **WEP**-Modell ist die bedingte Verzweigung (siehe Abbildung 7.3). Bedingte Verzweigungen bilden einen symmetrischen Block, der mehrere Aktivitäten, die in verschiedenen Strängen angeordnet sind, enthält. Anfang und Ende des Blocks sind durch eindeutige Start- und Endaktivitäten gegeben. Abhängig vom Rückgabewert (Returncode) der Startaktivität wird ein Strang der Verzweigung durchlaufen (1-of-n-split/1-of-n-join), wobei die anderen Stränge unberührt bleiben.

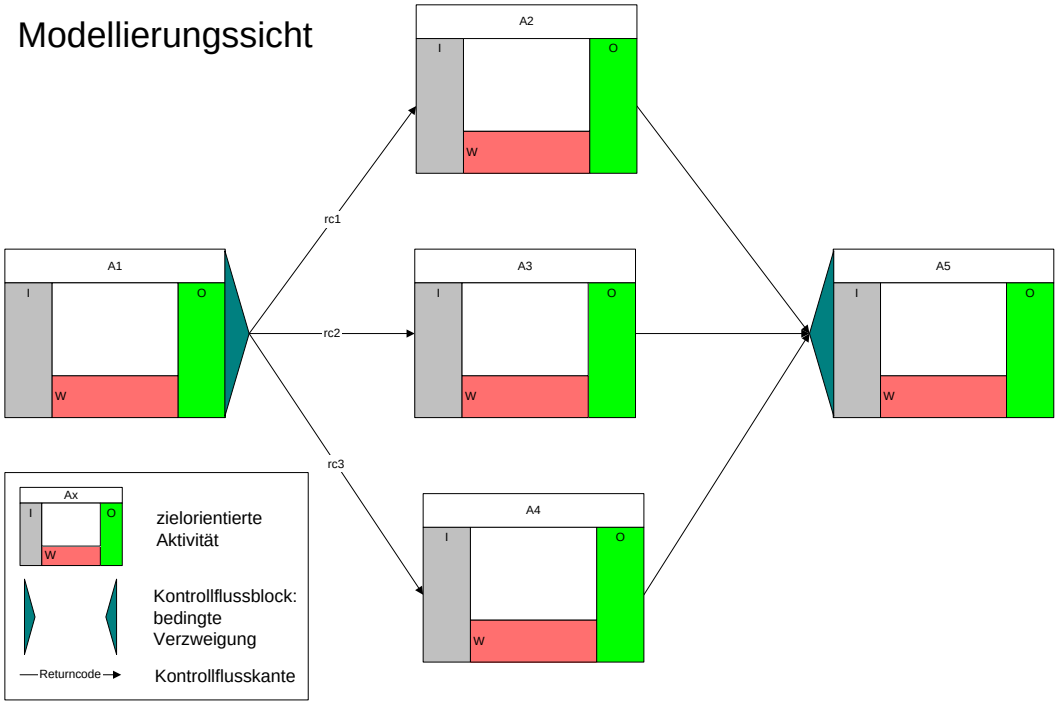



**Abbildung 7.3:** Bedingte Verzweigung im **WEP**-Modell

### 7.2.3 Modellierung von Schleifen

Auch Schleifen (siehe Abbildung 7.4) werden im **WEP**-Modell als ein Kontrollflusskonstrukt betrachtet. Die Schleife wird abhängig vom Rückgabewert (Returncode) der letzten Aktivität im Schleifenrumpf verlassen oder erneut durchlaufen.

### 7.2.4 Modellierung statischer Parallelität

Parallelität wird im **WEP**-Modell wie in herkömmlichen Workflow-Management-Systemen durch parallele Verzweigungen bereitgestellt (siehe Abbildung 7.5). Nach Beendigung der Startaktivität der parallelen Verzweigung werden die nachfolgenden Kontrollflussstränge parallel freigeschaltet (n-of-n-split/n-of-n-join).

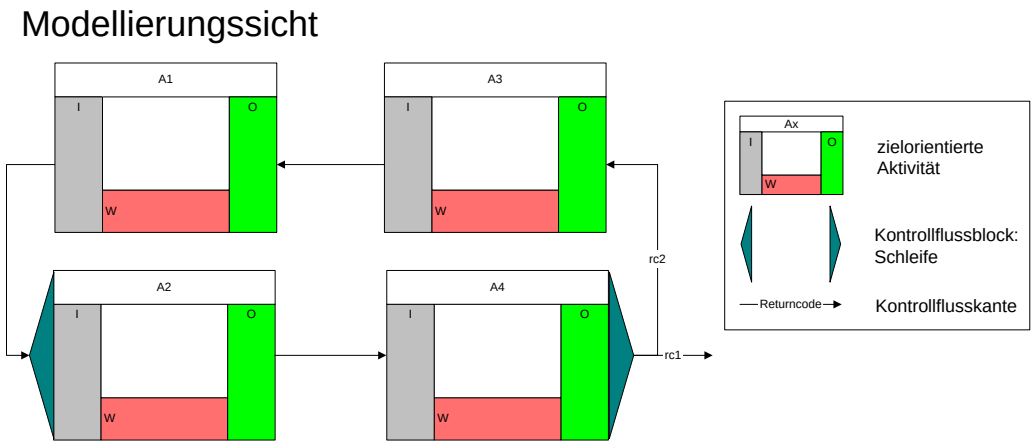



**Abbildung 7.4:** Schleifenkonstrukt im **WEP**-Modell

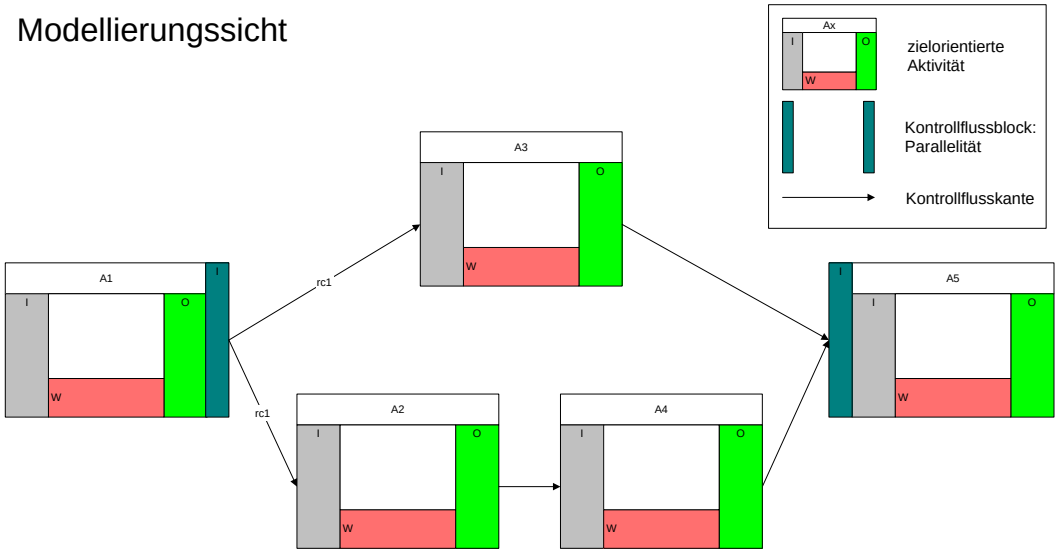



**Abbildung 7.5:** Statische Parallelität im **WEP**-Modell

### 7.2.5 Modellierung dynamischer Parallelität

Statische Parallelität reicht für die Modellierung von Produktentwicklungsprozessen für praktische Anwendungen nicht aus, weil die Anzahl der parallelen Zweige zur Modellierungszeit nicht festgelegt werden kann (vergleiche auch Teil I der Arbeit). Diese Anzahl der parallelen Zweige kann erst zur Laufzeit aus der konkreten Objektstruktur der im Workflow erzeugten Objektinstanzen abgeleitet werden. Beispielsweise kann bei einem Änderungsprozess die variable Anzahl von Einzelteilen nicht durch eine vorgefertigte Kontrollflussmodellierung dargestellt werden.

Um diesem Sachverhalt gerecht zu werden, gibt es im **WEP**-Modell das Konstrukt der *Traversierung*, das zur Beschreibung von Parallelität mit einer *variablen Anzahl paralleler Prozesszweige* dient. Bei der Traversierung wird ein komplex strukturiertes Eingabeobjekt zerlegt. Entsprechend der Anzahl der Subobjekte wird eine variable Anzahl von Subprozessen gestartet. Die Abbildungen 7.6 und 7.7 zeigen einen Ausschnitt eines **WEP**-Workflows mit Traversierungsmerkmal zur Modellierungs- und zur Laufzeit. Die Aktivitäten $A2$ und $A3$ sind von der Traversierung eingeschlossen und können zur Laufzeit in einer dynamischen Anzahl von Instanzen parallel gestartet werden. In der Abbildung wurden bei der Traversierung zur Laufzeit drei Subobjekte gefunden und dementsprechend drei Instanzen der beiden Aktivitäten erzeugt. Die Definition eines Traversierungsmerkmals im **WEP**-Modell wird im Detail nach Einführung des Datenflusses beschrieben.

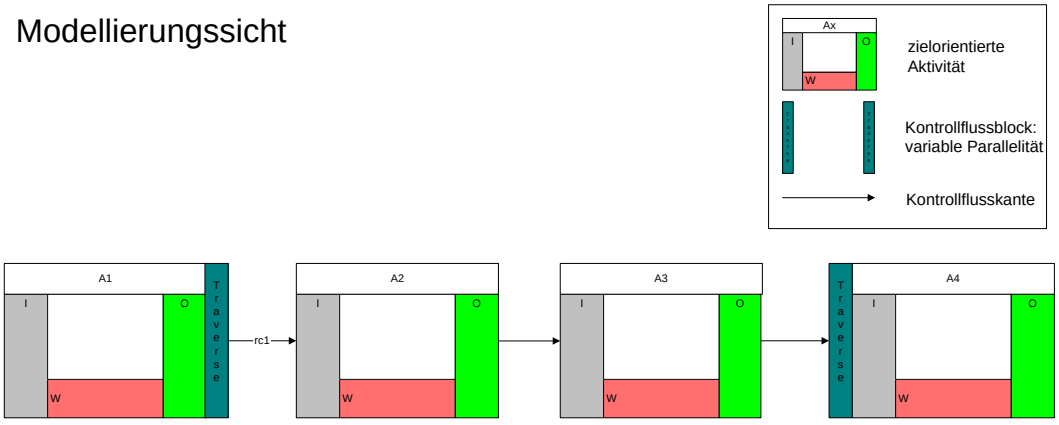



**Abbildung 7.6:** Dynamische Parallelität zur Modellierungszeit im **WEP**-Modell

## 7.3 Formale Spezifikation des Kontrollflusses

Formal lässt sich der **WEP**-Kontrollfluss mit seinen verschiedenen Kontrollflusskonstrukten durch die folgende Grammatik beschreiben:

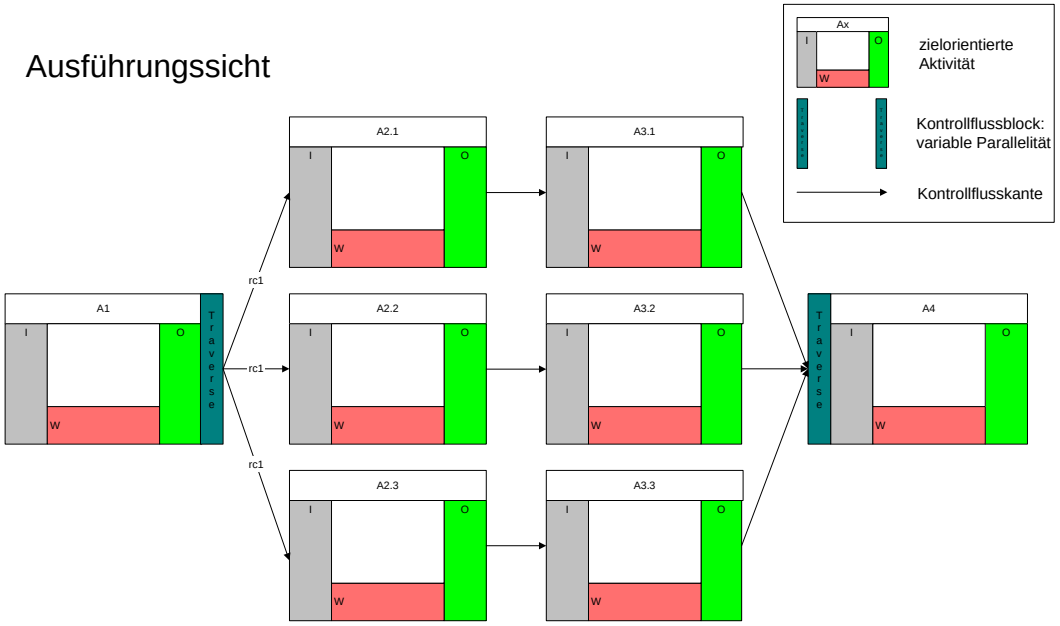



**Abbildung 7.7:** Dynamische Parallelität zur Laufzeit im **WEP**-Modell

$$
\begin{aligned}
ControlFlow &\rightarrow \langle_s\, CFblock\rangle_e \\
CFblock &\rightarrow \langle CFblockSpec\rangle \mid \langle\langle CFblockSpec\rangle - CFblock\rangle \\
CFblockSpec &\rightarrow Activity \mid CFaltBranch \mid CFstatparBranch \mid \\
&\quad CFdynparBranch \mid CFloop \\
CFaltBranch &\rightarrow Activity \blacktriangleright^{\prec} \{CFaltBranchSpec\}^{\succ} \blacktriangleleft Activity \\
CFstatparBranch &\rightarrow Activity \dashv^{st} \{CFstatparBranchSpec\} \,^{st}\vdash Activity \\
CFloop &\rightarrow \blacktriangleright^{\circlearrowleft} CFinnerLoopSpec^{\circlearrowleft} \blacktriangleleft CFexitSpec, CFoptionalLoopSpec \\
CFdynparBranch &\rightarrow Activity \dashv^{tv} \{CFdynparBranchSpec\} \,^{tv}\vdash Activity \\
CFaltBranchSpec &\rightarrow (Areturncode, CFblock) \mid \\
&\quad (Areturncode, CFblock), CFaltBranchSpec \\
CFstatparBranchSpec &\rightarrow CFblock \mid \\
&\quad CFblock, CFstatparBranchSpec \\
CFdynparBranchSpec &\rightarrow (TraverseFeature, CFblock) \mid \\
&\quad (TraverseFeature, CFblock), CFdynparBranchSpec \\
CFinnerLoopSpec &\rightarrow Activity \mid \\
&\quad Activity - Activity \mid \\
&\quad Activity - CFblock - Activity \\
CFoptionalLoopSpec &\rightarrow (Areturncode, CFblock) \mid (Areturncode) \\
CFexitSpec &\rightarrow (Areturncode)
\end{aligned}
$$

*$ControlFlow, CFblock$:*

Die in Abschnitt 7.1 eingeführte Blockstruktur spiegelt sich in den ersten beiden Regeln wider. Der gesamte Kontrollfluss eines **WEP**-Workflows wird durch die Terminale $\langle_s$ und $\rangle_e$ umschlossen. Dieser Block bildet eine endliche Sequenz (Terminalsymbol $-$) von (geschachtelten) Blöcken, wobei jede Sequenz aus einem beliebigen Kontrollflussblock (Nonterminal $CFblockSpec$) gefolgt von einer weiteren Sequenz von Blöcken besteht.

*$CFblockSpec$:*

Hiermit wird ein einzelner Kontrollflussblock beschrieben. Ein solcher Block kann sein:

- eine zielorientierte Aktivität: $Activity$. Die formale Beschreibung einer Aktivität erfolgte bereits in Abschnitt 6.2.
- eine bedingte Verzweigung $CFaltBranch$. Der Block einer bedingten Verzweigung beginnt mit einer Aktivität $A_{altBranch}$, über deren Returncode ein Zweig exklusiv ausgewählt wird, und endet mit einer Synchronisationsaktivität $A_{altSync}$. Die Zweige der bedingten Verzweigung werden als Menge von Tupeln aus Returncode und Kontrollflussblock ($Areturncode$, $CFblock$) beschrieben. Dabei muss gelten, dass für jeden Returncode der Aktivität $A_{altBranch}$ genau eines dieser Tupel existiert:

  Seien $CFaltBranch(WF)$ alle Kontrollflusskonstrukte vom Typ bedingter Verzweigung eines Workflows $WF$:

  $$\begin{aligned} CFaltBranch(WF) \;:=\; & \{w \in \mathbf{T}^*_{\mathbf{WEP}} : CFaltBranch \rightarrow^* w \wedge \\ & w\, IsPartialWordOf\, WF\} \end{aligned}$$

  Sei $cf_{altBranch}$ eine beliebige bedingte Verzweigung aus $CFaltBranch(WF)$ und sei

  $$\begin{aligned} A_{altBranch} \;:=\; & \{w \in \mathbf{T}^*_{\mathbf{WEP}} : Activity \rightarrow^* w \wedge \\ & w \blacktriangleright^{\prec} IsPartialWordOf\ cf_{altBranch} \wedge \\ & cf_{altBranch} \in CFaltBranch(WF)\} \end{aligned}$$

  die dazugehörende Verzweigungsaktivität. Dann gilt:

  $$\begin{aligned} \forall rc \in RETURNCODES(A_{altBranch})\ \exists_1 br, rc' \in \mathbf{T}^*_{\mathbf{WEP}} : \\ & Areturncode \rightarrow^* rc' \wedge \\ & CFaltBranchSpec \rightarrow^* br \wedge \\ & br\, IsPartialWordOf\ cf_{altBranch} \wedge \\ & rc'\, IsPartialWordOf\ br \wedge \\ & rc' == rc \end{aligned}$$

- eine statische Parallelität $CFstatparBranch$. Auch der Block einer statischen Parallelität beginnt mit einer Verzweigungsaktivität $A_{parBranch}$, die genau einen Returncode besitzt, und endet mit einer Synchronisationsaktivität $A_{parSync}$. Die Anzahl der parallelen Zweige ist frei. Jeder Zweig einer parallelen Verzweigung wird als ein beliebiger Block $CFblock$ repräsentiert.
- eine dynamische Parallelität $CFdynparBranch$. Der Block einer dynamischen Parallelität startet und endet ebenfalls mit einer Aktivität. Die Verzweigungsaktivität besitzt nur einen Returncode. Zur Laufzeit wird für jedes Traversierungsmerkmal immer der

gleiche Block dynamisch zu mehreren parallelen Zweigen vervielfacht. Die Zahl der parallelen Zweige wird durch die Anzahl der *Traversierungstreffer* bestimmt. Traversierungstreffer sind alle Subobjektversionen der zur Laufzeit traversierten Objektversion, welche die im Traversierungsmerkmal $TraverseFeature$ beschriebenen Eigenschaften vorweisen können. Die formale Darstellung von Traversierungsmerkmalen ist erst nach Einführung des Datenflusses (Abschnitt 8.2) möglich. Sie ist deshalb in Abschnitt 8.3.2 zu finden.

- eine Schleife $CFloop$. Zur Sicherstellung einer korrekten Blockstruktur muss der innere Block einer Schleife, der sicher einmal durchlaufen wird, mit einer Aktivität $A_{loopBegin}$ beginnen und einer Aktivität $A_{loopEnd}$ enden, wobei diese beiden Aktivitäten zu einer verschmelzen können. Zwischen diesen beiden Aktivitäten kann ein beliebiger Block eingefügt werden. Dieser innere Block wird von den Terminalen $\blacktriangleright^{\circlearrowleft}$ und ${}^{\circlearrowleft}\blacktriangleleft$ umschlossen. Zusätzlich wird eine Schleifenbedingung $CFoptionalLoopSpec$, die aus Returncode und optionalem Kontrollflussblock besteht, und eine Bedingung $CFexitSpec$ zum Verlassen der Schleife spezifiziert. Die letzte Aktivität $A_{loopEnd}$ des inneren Blockes besitzt genau zwei Returncodes, die auf die Bedingungen $CFoptionalLoopSpec$ und $CFexitSpec$ abgebildet werden:

  Seien $CFloop(WF)$ alle Schleifen-Kontrollflusskonstrukte eines Workflows $WF$:

$$\begin{aligned} CFloop(WF) \quad := \quad & \{w \in \mathbf{T}^{*}_{\mathbf{WEP}} : CFloop \rightarrow^{*} w \wedge \\ & w \; IsPartialWordOf \; WF\} \end{aligned}$$

  Sei $cf_{loop}$ eine beliebige Schleife aus $CFloop(WF)$ und sei

$$\begin{aligned} A_{loopEnd} \quad := \quad & \{w \in \mathbf{T}^{*}_{\mathbf{WEP}} : Activity \rightarrow^{*} w \wedge \\ & w^{\circlearrowleft}\blacktriangleleft IsPartialWordOf \; cf_{loop} \wedge \\ & cf_{loop} \in CFloop(WF)\} \end{aligned}$$

  die dazugehörende Verzweigungsaktivität, also die letzte Aktivität des inneren Blocks. Dann gilt:

$$\begin{aligned} & |RETURNCODES(A_{loopEnd})| = 2 \\ \exists_1 rc_{loop} \in RETURNCODES(A_{loopEnd}) : \; & Areturncode \rightarrow^{*} rc_{loop} \wedge \\ & CFoptionalLoopSpec \rightarrow^{*} rc_{loop} \wedge \\ & rc_{loop} \; IsPartialWordOf \; cf_{loop} \\ \exists_1 rc_{exit} \in RETURNCODES(A_{loopEnd}) : \; & Areturncode \rightarrow^{*} rc_{exit} \wedge \\ & CFexitSpec \rightarrow^{*} rc_{exit} \wedge \\ & rc_{exit} \; IsPartialWordOf \; cf_{loop} \end{aligned}$$

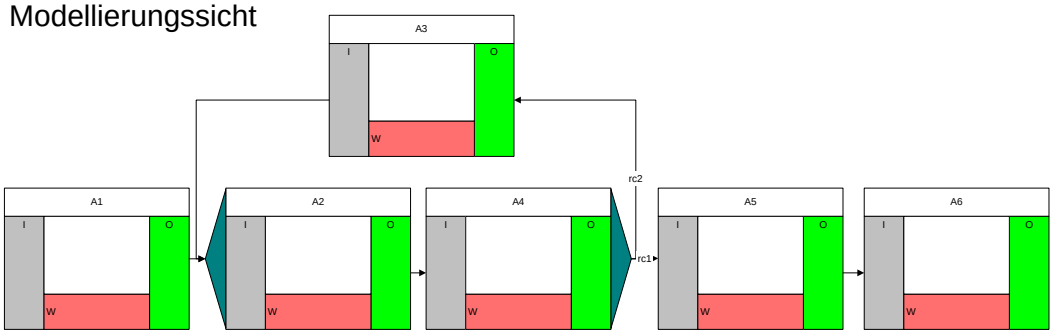



**Abbildung 7.8:** Beispiel einer Schleife im **WEP**-Modell

*Beispiel 3: Schleife: Rechtsableitung des Kontrollflusses aus Abbildung 7.8*

$$
\begin{aligned}
&ControlFlow \rightarrow \langle_s\, CFblock\rangle_e\\
&\rightarrow \langle_s\, \langle\langle CFblockSpec\rangle - CFblock\rangle\rangle_e\\
&\rightarrow \langle_s\, \langle\langle CFblockSpec\rangle - \langle\langle CFblockSpec\rangle - CFblock\rangle\rangle\rangle_e\\
&\rightarrow \langle_s\, \langle\langle CFblockSpec\rangle - \langle\langle CFblockSpec\rangle - \langle\langle CFblockSpec\rangle - CFblock\rangle\rangle\rangle\rangle_e\\
&\rightarrow \langle_s\, \langle\langle CFblockSpec\rangle - \langle\langle CFblockSpec\rangle - \langle\langle CFblockSpec\rangle - \langle CFblockSpec\rangle\rangle\rangle\rangle\rangle_e\\
&\rightarrow \langle_s\, \langle\langle CFblockSpec\rangle - \langle\langle CFblockSpec\rangle - \langle\langle CFblockSpec\rangle - \langle Activity\rangle\rangle\rangle\rangle\rangle_e\\
&\rightarrow \langle_s\, \langle\langle CFblockSpec\rangle - \langle\langle CFblockSpec\rangle - \langle\langle CFblockSpec\rangle - \langle(A6)\rangle\rangle\rangle\rangle\rangle_e\\
&\rightarrow \langle_s\, \langle\langle CFblockSpec\rangle - \langle\langle CFblockSpec\rangle - \langle\langle Activity\rangle - \langle(A6)\rangle\rangle\rangle\rangle\rangle_e\\
&\rightarrow \langle_s\, \langle\langle CFblockSpec\rangle - \langle\langle CFblockSpec\rangle - \langle\langle(A5)\rangle - \langle(A6)\rangle\rangle\rangle\rangle\rangle_e\\
&\rightarrow \langle_s\, \langle\langle CFblockSpec\rangle - \langle\langle CFloop\rangle - \langle\langle(A5)\rangle - \langle(A6)\rangle\rangle\rangle\rangle\rangle_e\\
&\rightarrow \langle_s\, \langle\langle CFblockSpec\rangle - \langle\langle \blacktriangleright^{\circlearrowleft} CFinnerLoopSpec^{\circlearrowleft} \blacktriangleleft CFexitSpec, CFoptionalLoopSpec\rangle -\\
&\quad \langle\langle(A5)\rangle - \langle(A6)\rangle\rangle\rangle\rangle\rangle_e\\
&\rightarrow \langle_s\, \langle\langle CFblockSpec\rangle - \langle\langle \blacktriangleright^{\circlearrowleft} CFinnerLoopSpec^{\circlearrowleft} \blacktriangleleft CFexitSpec, (Areturncode, CFblock)\rangle -\\
&\quad \langle\langle(A5)\rangle - \langle(A6)\rangle\rangle\rangle\rangle\rangle_e\\
&\rightarrow \langle_s\, \langle\langle CFblockSpec\rangle - \langle\langle \blacktriangleright^{\circlearrowleft} CFinnerLoopSpec^{\circlearrowleft} \blacktriangleleft CFexitSpec, (Areturncode, \langle CFblockSpec\rangle)\rangle -\\
&\quad \langle\langle(A5)\rangle - \langle(A6)\rangle\rangle\rangle\rangle\rangle_e\\
&\rightarrow \langle_s\, \langle\langle CFblockSpec\rangle - \langle\langle \blacktriangleright^{\circlearrowleft} CFinnerLoopSpec^{\circlearrowleft} \blacktriangleleft CFexitSpec, (Areturncode, \langle Activity\rangle)\rangle -\\
&\quad \langle\langle(A5)\rangle - \langle(A6)\rangle\rangle\rangle\rangle\rangle_e\\
&\rightarrow \langle_s\, \langle\langle CFblockSpec\rangle - \langle\langle \blacktriangleright^{\circlearrowleft} CFinnerLoopSpec^{\circlearrowleft} \blacktriangleleft CFexitSpec, (Areturncode, \langle(A3)\rangle)\rangle -\\
&\quad \langle\langle(A5)\rangle - \langle(A6)\rangle\rangle\rangle\rangle\rangle_e\\
&\rightarrow \langle_s\, \langle\langle CFblockSpec\rangle - \langle\langle \blacktriangleright^{\circlearrowleft} CFinnerLoopSpec^{\circlearrowleft} \blacktriangleleft CFexitSpec, ((rc2), \langle(A3)\rangle)\rangle -\\
&\quad \langle\langle(A5)\rangle - \langle(A6)\rangle\rangle\rangle\rangle\rangle_e\\
&\rightarrow \langle_s\, \langle\langle CFblockSpec\rangle - \langle\langle \blacktriangleright^{\circlearrowleft} CFinnerLoopSpec^{\circlearrowleft} \blacktriangleleft (Areturncode), ((rc2), \langle(A3)\rangle)\rangle -\\
&\quad \langle\langle(A5)\rangle - \langle(A6)\rangle\rangle\rangle\rangle\rangle_e\\
&\rightarrow \langle_s\, \langle\langle CFblockSpec\rangle - \langle\langle \blacktriangleright^{\circlearrowleft} CFinnerLoopSpec^{\circlearrowleft} \blacktriangleleft (rc1), ((rc2), \langle(A3)\rangle)\rangle - \langle\langle(A5)\rangle - \langle(A6)\rangle\rangle\rangle\rangle\rangle_e\\
&\rightarrow \langle_s\, \langle\langle CFblockSpec\rangle - \langle\langle \blacktriangleright^{\circlearrowleft} Activity - Activity^{\circlearrowleft} \blacktriangleleft (rc1), ((rc2), \langle(A3)\rangle)\rangle - \langle\langle(A5)\rangle - \langle(A6)\rangle\rangle\rangle\rangle\rangle_e\\
&\rightarrow \langle_s\, \langle\langle CFblockSpec\rangle - \langle\langle \blacktriangleright^{\circlearrowleft} Activity - (A4)^{\circlearrowleft} \blacktriangleleft (rc1), ((rc2), \langle(A3)\rangle)\rangle - \langle\langle(A5)\rangle - \langle(A6)\rangle\rangle\rangle\rangle\rangle_e\\
&\rightarrow \langle_s\, \langle\langle CFblockSpec\rangle - \langle\langle \blacktriangleright^{\circlearrowleft} (A2) - (A4)^{\circlearrowleft} \blacktriangleleft (rc1), ((rc2), \langle(A3)\rangle)\rangle - \langle\langle(A5)\rangle - \langle(A6)\rangle\rangle\rangle\rangle\rangle_e\\
&\rightarrow \langle_s\, \langle\langle Activity\rangle - \langle\langle \blacktriangleright^{\circlearrowleft} (A2) - (A4)^{\circlearrowleft} \blacktriangleleft (rc1), ((rc2), \langle(A3)\rangle)\rangle - \langle\langle(A5)\rangle - \langle(A6)\rangle\rangle\rangle\rangle\rangle_e\\
&\rightarrow \langle_s\, \langle\langle(A1)\rangle - \langle\langle \blacktriangleright^{\circlearrowleft} (A2) - (A4)^{\circlearrowleft} \blacktriangleleft (rc1), ((rc2), \langle(A3)\rangle)\rangle - \langle\langle(A5)\rangle - \langle(A6)\rangle\rangle\rangle\rangle\rangle_e
\end{aligned}
$$

# Kapitel 8

# Datenmanagement: Datenflussmodellierung und globale Objekte

Betrachtet man heutige Workflow-Management-Systeme, so bieten sie häufig keine explizite Datenflussmodellierung an (folder-orientierte Systeme [VE92, KR91, BMR94, SNI95], petrinetzbasierte Systeme [Obe94b]). Existiert eine explizite Datenflussmodellierung, so beschränkt sich diese auf einfache Datentypen [LA94, Rei00].

Für eine konsistente Workflow-Modellierung ist eine explizite Beschreibung des Datenflusses zwischen den Prozessschritten eines Workflows unabdingbar. Nur dadurch können Vollständigkeit und Korrektheit zur Modellierungszeit überprüft werden [RBFD01], so dass man zur Laufzeit vor unliebsamen Überraschungen, wie dem Blockieren von Workflow-Instanzen, sicher ist. Eine vollständige und korrekte Datenflussmodellierung liegt dann vor, wenn für jeden Prozessschritt eines Workflows beim Start alle notwendigen Eingabeparameter vorliegen, unabhängig vom gewählten Kontrollflusspfad durch den Workflow-Graphen.

Bei den heutigen Systemen fehlt eine weitreichende methodische Unterstützung bei der Datenflussmodellierung. Eine Validierung einer korrekten Parameterversorgung und -weiterleitung wird nicht angeboten. Stattdessen bleibt es den Workflow-Modellierern überlassen, dafür zu sorgen, dass zur Ausführungszeit eine korrekte Datenversorgung gewährleistet ist. Einige wenige Ansätze erlauben die Simulation und Animation möglicher Datenflüsse [Rol96, Obe94b].

Für komplexere Workflow-Anwendungen ist diese Vorgehensweise nicht adäquat. Anwendungen aus der Praxis zeigen, dass ohne systemseitige Unterstützung eine korrekte Datenflussbeschreibung sehr aufwändig und fehleranfällig ist. Ein Blockieren von Workflows zur Laufzeit aufgrund fehlender oder unvollständiger Daten kann nicht ausgeschlossen werden. Um diese Gefahr zu reduzieren, werden aufwändige Testszenarien in praxisnahen Pilotanwendungen gefahren, bevor die modellierten Workflows in den eigentlichen Produktivsystemen freigegeben werden.

Wie bereits in der Einleitung aufgezeigt, besitzen Entwicklungsprozesse eine komplexe Kontroll- und Datenflussstruktur. Durch die Forderung nach vorzeitiger Datenweitergabe kommt erschwerend hinzu, dass bei der Verknüpfung von Ein- und Ausgabeparametern unterschiedliche Datenqualitäten zu

berücksichtigen sind und dass Ein- und Ausgabeobjekte komplexe Strukturen aufweisen. Zusätzlich werden durch Traversierung komplexer Objektstrukturen zur Laufzeit sowohl Kontroll- als auch Datenflüsse dynamisch verändert (vergleiche Abschnitt 7.2.5 und Kapitel 16). Analysen von Entwicklungsprozessen haben darüber hinaus gezeigt, dass auch zwischen parallelen Zweigen der Austausch von Daten benötigt wird. Das übliche Vorgehen der Modellierung künstlicher Synchronisationsschritte, um Datenflüsse zwischen parallelen Zweigen zu vermeiden, kann zu erheblichen zeitlichen Verzögerungen der Gesamtprozesslaufzeit führen (vergleiche Abbildung 8.1), so dass diese Vorgehensweise für Entwicklungsprozesse nicht angewendet werden kann.

Es ist im **WEP**-Modell deshalb unabdingbar, dass Datenflussmodellierer mittels geeigneter Datenflussregeln geführt werden, so dass nur korrekte Datenflussverknüpfungen zugelassen werden. Diese Regeln müssen sowohl Datenqualitäten als auch Traversierungsrichtlinien berücksichtigen. Die Basis bildet auch hier wieder ein formales Modell, das schrittweise eingeführt wird.

## 8.1 Globale Objekte zur Repräsentation von Produktdaten

Wie bereits in Kapitel 5 erwähnt, basiert die Modellierung von **WEP**-Workflows idealerweise auf einem importierten, komplexen Datenmodell, das Produktdaten beschreibt. Zur Laufzeit werden die in einem **WEP**-Workflow benötigten Produktdaten nicht im **WEP**-Workflow-Management-System, sondern in externen Datenverwaltungssystemen, sogenannten *Produktdatenmanagementsystemen* verwaltet. Um den Zugriff auf die im Workflow benötigten Datenobjekte zu ermöglichen, muss im **WEP**-Workflow ein geeigneter *Repräsentant* des Objekts im Produktdatenmanagementsystem modelliert werden. Diese Repräsentanten werden im **WEP**-Modell *globale Objekte* genannt. Sie nehmen zur Laufzeit (Zwischen-)Ergebnisse entgegen und leiten sie über Datenflusskanten an Folgeaktivitäten weiter. Sie verwalten damit alle *Objektversionen* eines Objekts.

Jedes globale Objekt besitzt einen workflow-weiten eindeutigen Namen und eine Klasse. Die formale Spezifikation lautet:

$$GlobalObject \quad \rightarrow \quad (GOname, GOclass)$$

$GOname$*:*
: $GOname \in UNIQUE$ beschreibt den eindeutigen Namen des globalen Objekts.

$GOclass$*:*
: $GOclass \in Oclass$ ordnet $GOname$ eine bereits vorhandene Objektklasse zu.

Damit lässt sich die Menge aller globalen Objekte eines beliebigen **WEP**-Workflows $WF \in$ **WEP**-$WORKFLOWS$ als Teilmenge der Menge aller erlaubten Wörter der **WEP**-Sprache $\mathbf{T}^*_{\mathbf{WEP}}$ definieren:

$$GLOBALOBJECTS(WF) := \{O \in \mathbf{T}^*_{\mathbf{WEP}} : GlobalObject \rightarrow^* O \wedge O\ IsPartialWordOf\ WF\}$$

## 8.2 Datenflussmodellierung

Der Datenfluss zwischen Aktivitäten eines **WEP**-Workflows $WF$ wird modelliert, indem ihre Ein- und Ausgabeparameter über Datenflusskanten mit den globalen Objekten des Workflows $WF$ ver-

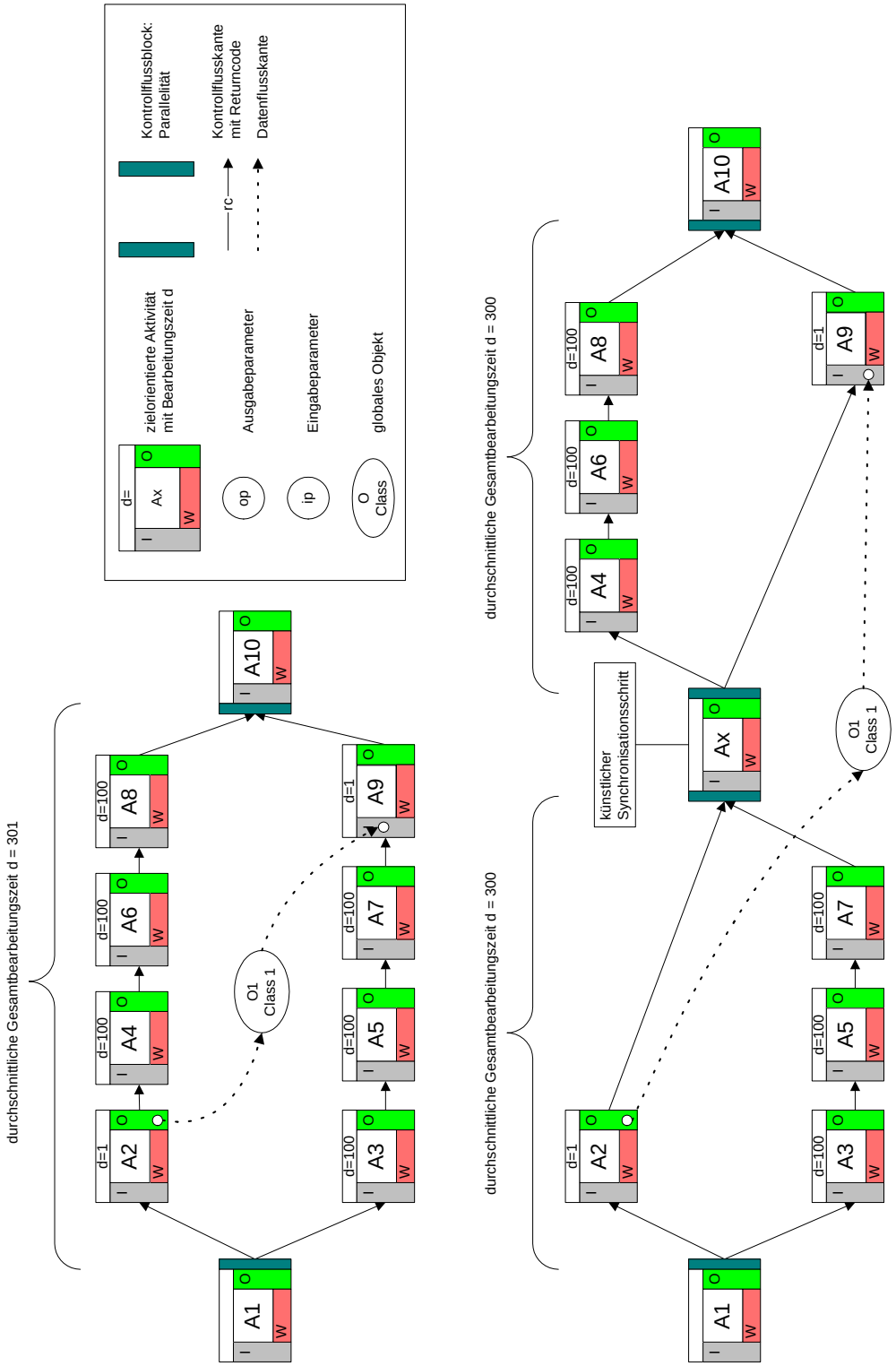



**Abbildung 8.1:** Beispiel für die Notwendigkeit des Datenflusses zwischen parallelen Zweigen: Die unterschiedlich langen Bearbeitungszeiten (d=. . . ) führen fast zur Verdopplung der Prozessdurchlaufzeiten

knüpft werden. Eine Datenflusskante im **WEP**-Modell verbindet also entweder einen Ausgabeparameter einer Aktivität mit einem globalen Objekt (*Datenflussausgangskante*) oder ein globales Objekt mit einem Aktivitäteneingabeparameter (*Datenflusseingangskante*).

$$
\begin{aligned}
DataFlow &\rightarrow \{DFspec\} \\
DFspec &\rightarrow DFinSpec, DFspec \mid DFoutSpec, DFspec \mid DFinSpec \mid DFoutSpec \\
DFinSpec &\rightarrow (GlobalObject, Activity, APIspec) \\
DFoutSpec &\rightarrow (Activity, APOspec, GlobalObject)
\end{aligned}
$$

$DataFlow$, $DFspec$:
: Der Datenfluss ist eine Menge aller Datenflusseingangskanten $DFinSpec$ und Datenflussausgangskanten $DFoutSpec$.

$DFinSpec$:
: Eine Datenflusseingangskante beginnt bei einem globalen Objekt $GlobalObject$ und endet beim Eingabeparameter $APIspec$ einer Aktivität. Es muss gelten, dass $APIspec \in APINPUTS(Activity)$ ist. Sie wird formal als Tripel der Form ($GlobalObject$, $Activity$, $APIspec$) dargestellt.

$DFoutSpec$:
: Eine Datenflussausgangskante beginnt beim Ausgabeparameter $APOspec$ einer Aktivität und endet bei einem globalen Objekt $GlobalObject$. Es muss gelten, dass $APOspec \in APOUTPUTS(Activity)$ ist. Sie wird formal als Tripel der Form ($Activity$, $APOspec$, $GlobalObject$) dargestellt.

Die Menge aller Datenflusskanten eines Workflows $WF \in$ **WEP**-$WORKFLOWS$ wird mit $DATAFLOW(WF)$ bezeichnet. Sie wird als Teilmenge der Menge aller erlaubten Wörter der **WEP**-Sprache $\mathbf{T}^*_{\mathbf{WEP}}$ definiert:

$$DATAFLOW(WF) := \{dfe \in \mathbf{T}^*_{\mathbf{WEP}} : DataFlow \rightarrow^* dfe \ \wedge dfe \, IsPartialWordOf \ WF\}$$

Der mit diesen Kanten aufgespannte Graph wird das *Datenflussmodell* des Workflows $WF$ genannt.

## 8.3 Automatische dynamische Prozessaufsplittung

Das Kontrollflusskonstrukt der *dynamischen Parallelität* (vergleiche Abschnitt 7.2.5) ermöglicht eine parallele Aufsplittung des Workflow-Graphen zur Laufzeit, indem eine komplexe Objektversion bei ihrer Weitergabe entsprechend vorgegebener Regeln (siehe *Traversierungsmerkmal*, Abschnitt 8.3.2) traversiert, die Subobjektversionen auf Eigenschaften überprüft und bei jedem Erfüllen dieser Eigenschaften (*Traversierungstreffer*) dynamisch ein neuer paralleler Zweig erzeugt wird.

Im Folgenden wird das im Abschnitt 7.2.5 eingeführte Modellierungskonstrukt der dynamischen Parallelität um die noch fehlenden Datenflussaspekte ergänzt und danach im Detail die Definition von Traversierungsmerkmalen beschrieben.

### 8.3.1 Datenflussaspekte bei der Modellierung dynamischer Parallelität

Ziel der dynamischen Parallelität ist es, bei einem komplex strukturierten Objekt $OTV$ zur Laufzeit Subobjekte mit bestimmten Eigenschaften zu finden, jedes Subobjekt, auf das diese Eigenschaften zutreffen (*Traversierungstreffer*), durch einen dynamisch erzeugten parallelen Zweig zu manipulieren und am Ende des Parallelitätsblocks die Subobjekte zum strukturierten Objekt $OTV$ wieder zusammenzubauen. Dabei werden sich in der Regel die Qualitätsstufen der Subobjekte und damit auch die des traversierten Objekts $OTV$ verändern.

Ausgangspunkt bei der Modellierung einer dynamischen Parallelität ist die Festlegung der Objektklassen des zu traversierenden globalen Objekts und dessen Qualitätsstufen $oql_{tvstart_1}$,..., $oql_{tvstart_n}$, deren Eintreffen die Traversierung auslösen soll. Außerdem muss spezifiziert werden, welche Qualitätsstufe $oql_{tvend_1}$, ..., $oql_{tvend_n}$ das traversierte Objekt beim „Zusammenbau“ am Ende des Parallelitätsblocks besitzen soll. Durch Vergleich der jeweiligen Qualitätsstufen kann nun analysiert werden, welche Subobjekte in anderen Qualitätsstufen bereitgestellt, welche Subobjekte innerhalb des Parallelitätsblocks neu erzeugt und welche Subobjekte nicht mehr benötigt werden. Für die ersten beiden Fälle wird zur Modellierungszeit jeweils ein Aktivitätenblock erzeugt, dessen Ausgabeparameter ein Objekt der Klasse des Subobjekts in der geforderten Qualitätsstufe erzeugt. Außerdem wird ein globales Objekt generiert, mit dem der Ausgabeparameter über eine Datenflussausgangskante verbunden wird.

Die Abbildungen 8.2 und 8.3 zeigen das Vorgehen für ein Qualitätsstufenpaar $oql_{tvstart_i}$, $oql_{tvend_i}$ exemplarisch: Die Traversierung wird angestoßen, wenn bei einem Objekt der Klasse $cls1$ die Qualitätsstufe $oqlX$ erreicht ist. Am Ende dieses dynamischen Parallelitätsblocks soll das Objekt die Qualitätsstufe $oqlY$ erreicht haben. Auf dem Weg dorthin müssen Subobjekte der Klasse $cls2$ von der Qualitätsstufe $oqlA$ auf die Qualitätsstufe $oqlC$ transferiert werden. Dazu wird ein Block $B1$ und ein Traversierungsmerkmal $oqlX.TV1$ modelliert. Der Block $B1$ führt die durch das Traversierungsmerkmal $oqlX.TV1$ spezifizierte Traversierungstreffer von der Qualitätsstufe $oqlA$ in die Qualitätsstufe $oqlC$ über. Subobjekte der Klasse $cls4$ werden nur in der Qualitätsstufe $oqlY$ der Klasse $cls1$ benötigt. Auch dafür muss ein Block $B2$ und ein Traversierungsmerkmal $oqlX.TV2$ erzeugt werden, der die Subobjekte in den entsprechenden Qualitätsstufen erzeugt. Außerdem müssen globale Objekte der entsprechenden Klassen generiert werden. Das Ergebnis ist in Abbildung 8.3 zu sehen. Abbildung 8.4 zeigt eine mögliche Ausprägung der dynamischen Parallelität zur Laufzeit.

Zwei Qualitätsstufen einer Objektklasse können sich nicht nur aufgrund unterschiedlicher Subobjektklassen beziehungsweise derer Qualitätsstufen unterscheiden, sondern auch aufgrund unterschiedlicher Attributbelegungen. Da innerhalb einer dynamischen Parallelität nur Subobjekte manipuliert werden können, können Attributbelegungen des traversierten Objekts nicht verändert werden. Dies muss bei der Wahl der Qualitätsstufen $oql_{tvstart_i}$, $oql_{tvend_i}$ berücksichtigt werden. Alternativ können die fehlenden Attributbelegungen in der mit dem Traversierungsende ($^{tv}$▮) assoziierten Aktivität nachgereicht werden.

### 8.3.2 Modellierung von Traversierungsmerkmalen

Natürlich darf eine Traversierung einer weitergegebenen Objektversion nicht aufgrund fehlender Traversierungstreffer zu einem Blockieren eines **WEP**-Workflows führen. Da jede an ein Traversierungsmerkmal weitergegebene Objektversion der qObjekt-Spezifikation eines Meilensteins der weiterge-

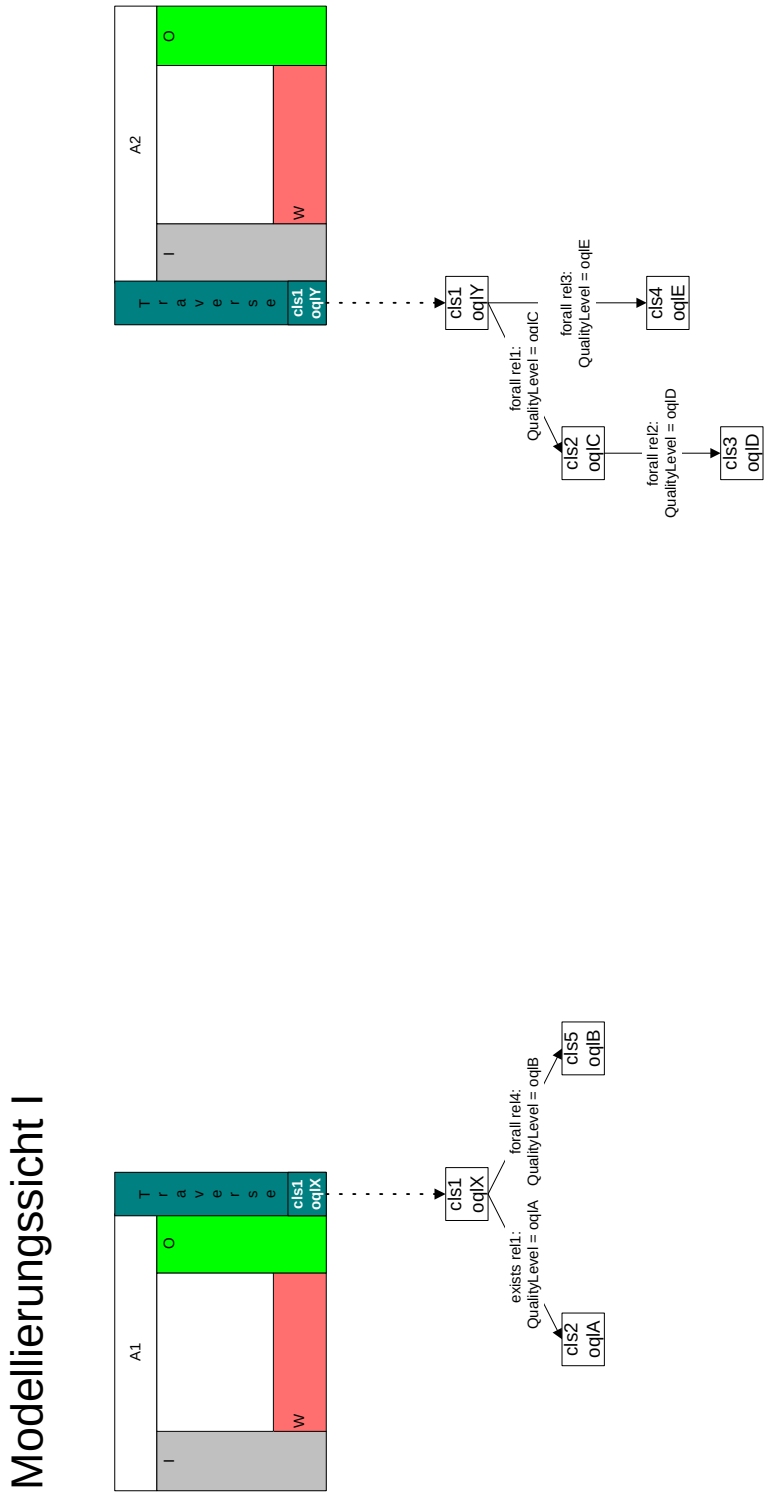



**Abbildung 8.2:** Datenflussaspekte bei der Modellierung dynamischer Parallelität: Analyse der Startqualitätsstufe und der Zielqualitätsstufe der zu traversierenden Objektklasse

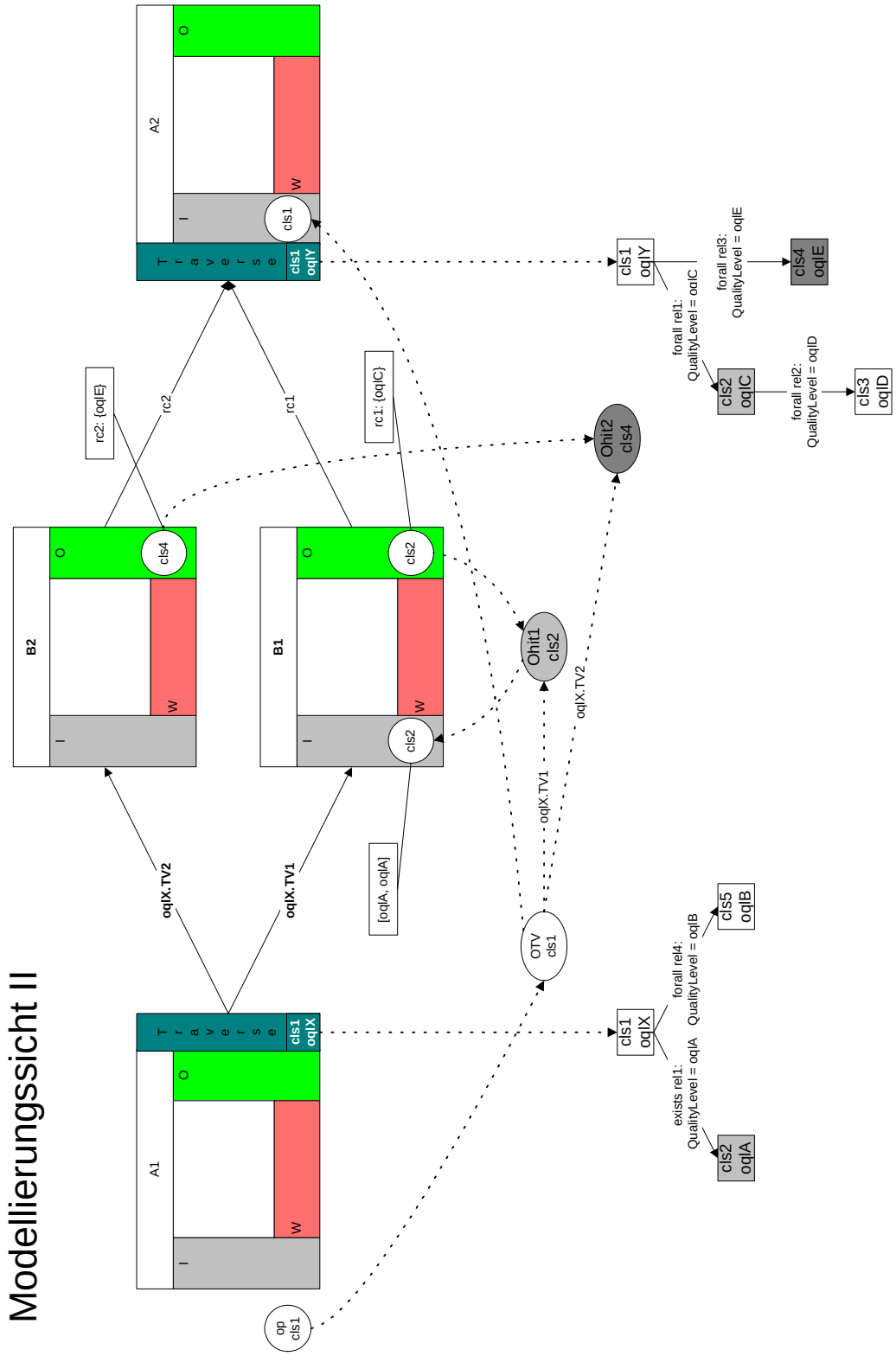



**Abbildung 8.3:** Datenflussaspekte bei der Modellierung dynamischer Parallelität: Analyse ergibt zwei Blöcke, zwei Traversierungsmerkmale und drei globale Objekte

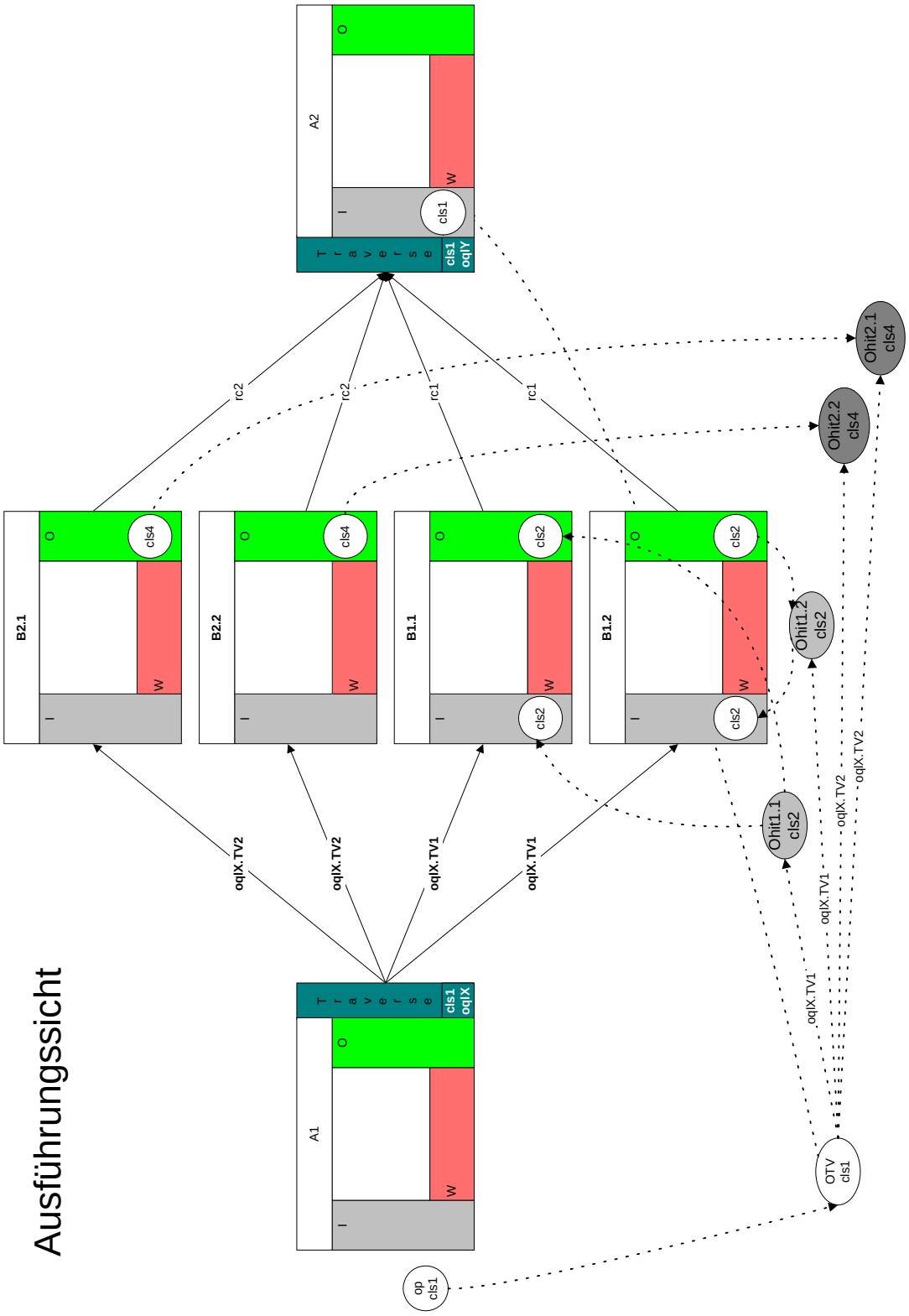



**Abbildung 8.4:** Mögliche Ausführungsszenario für das Beispiel aus Abbildung 8.3. Beide Traversierungsmerkmale führten zu zwei Traversierungstreffern.

benden Aktivität entspricht, werden sicherlich keine Traversierungstreffer erzielt, wenn sich diese qObjekt-Spezifikation und Traversierungsmerkmal widersprechen. Um dies im **WEP**-Modell auszuschließen, basiert die Modellierung eines Traversierungsmerkmals immer auf der Spezifikation eines qObjekts des Meilensteins. Ein Traversierungsmerkmal stellt damit eine *Verschärfung* der entsprechenden Kriterien eines Meilensteins dar.

Eine verschärfte Spezifikation liegt immer dann vor, wenn:

- bei einem Attribut statt der Forderung nach einer beliebigen Belegung ($= defined$) ein bestimmter Wert oder eine Menge bestimmter Werte verlangt wird.
- bei einer Relation statt des Existenz- der All-Quantor gefordert wird.
- weitere Attributbelegungen oder Relationen spezifiziert werden.

Hält ein Traversierungsmerkmal diese Bedingungen ein, so stellt die Menge aller dem Traversierungsmerkmal konformen Objektinstanzen eine Untermenge der Menge aller Objektinstanzen dar, die der entsprechenden Meilenstein-Spezifikation entsprechen.

Diese Modellierungsregel für Traversierungsmerkmale ist aus Anwendersicht sinnvoll, weil Bearbeiter von Aktivitäten bestrebt sind, ihre Meilensteine und damit das Erfüllen der assoziierten qObjekt-Spezifikationen zu erreichen. Ist es bei der Festlegung eines Traversierungsmerkmals notwendig, stark von diesen Vorgaben abzuweichen, so ist aus Modellierungssicht zu überlegen, diese Abweichungen (teilweise) in die Meilenstein-Beschreibung der betroffenen Aktivitäten zu übernehmen. Sie sind dann für Aktivitätenbearbeiter transparenter und damit leichter nachvollziehbar.

Die Modellierungsregel für Traversierungsmerkmale ist eine notwendige Voraussetzung für Traversierungstreffer zur Laufzeit. Sie ist aber nicht hinreichend, um ein Blockieren des Workflows aufgrund fehlender Eingabedaten zu verhindern. Der Workflow blockiert trotz korrekt modelliertem Traversierungsmerkmal zur Laufzeit, wenn die im Datenfluss zuletzt in das zu traversierende Objekt schreibende Aktivität beendet wird, ohne eine Objektversion weiterzugeben, die zu Traversierungstreffern führt. Dies ist immer dann der Fall, wenn die weitergegebenen Objektversionen die verschärften Kriterien des Traversierungsmerkmals nie erfüllen. Diese Überprüfung kann natürlich nicht zur Modellierungszeit, sondern muss zur Laufzeit erfolgen. Die Alternativen hierbei sind, dass bei jeder Weitergabe einer Objektversion überprüft wird, ob eine Traversierung mindestens einen Treffer enthält oder dass das Beenden einer Aktivität verhindert wird, wenn bei der letzten weitergegebenen Objektversion kein Traversierungstreffer vorhanden war.

Jeder Traversierungstreffer wird einem dynamisch erzeugten parallelen Zweig zugewiesen, der zur Modellierungszeit diesem Traversierungsmerkmal zugeordnet wurde. Alle parallelen Zweige beschreiben den gleichen Subprozess. Um Doppelarbeit zu verhindern muss sichergestellt werden, dass alle Traversierungstreffer auch auf unterschiedliche Objektversionen beziehungsweise auf Subobjektversionen einer bereits markierten Objektversion zeigen. Der erste Fall kann eintreten, weil dieselbe Subobjektversion über unterschiedliche Pfade (=Relationsbeziehungen) erreicht werden kann (vergleiche in Abbildung 5.3 die Subobjektversion $Part7$). Der andere Fall ist bei rekursiven Strukturen möglich. Ein Beispiel hierfür ist die Modul-Submodul-Beziehung über die Relation *consistsOfSubMods* im Datenmodell der Abbildung 5.1.

Der erste Fall kann nur zur Laufzeit ausgeschlossen werden, indem eine bereits als Traversierungstreffer markierte Objektversion bei einem weiteren Treffer nicht erneut einem parallelen Zweig zugeordnet wird.

Der zweite Fall wird in der nun folgenden formalen Spezifikation eines Traversierungsmerkmals dadurch ausgeschlossen, dass bei jedem Pfad eines Traversierungsmerkmals spezifiziert wird, ob die erste oder die letzte Objektversion des Pfades, die den Kriterien entspricht, als Treffer zurückgegeben wird.

$$
\begin{aligned}
TraverseFeature &\rightarrow (Tname, TrootqObjSpec, TleafObjSpec, \{TpathSet\}) \\
TrootqObjSpec &\rightarrow (Oclass, OqLevel) \mid (Oclass, OqLevel, TabstrBoolExpr) \\
TleafObjSpec &\rightarrow (Oclass) \\
TpathSet &\rightarrow (ThitPolicy, Tpath) \mid (ThitPolicy, Tpath), TpathSet \\
Tpath &\rightarrow (TtravPolicy, TpathSequence) \mid \\
&\quad\ (TtravPolicy, TpathSequence) - Tpath \\
TpathSequence &\rightarrow (TpathSectionSpec - TpathSequence) \mid (TpathSectionSpec) \\
TpathSectionSpec &\rightarrow (Trelation) \mid (Trelation, TabstrBoolExpr)
\end{aligned}
$$

*$TraverseFeature$:*

$TraverseFeature$ ist das Startsymbol zur formalen Beschreibung eines Traversierungsmerkmals. Ein Traversierungsmerkmal ist ein Tupel bestehend aus einem eindeutigen Namen $Tname \in UNIQUE$, der Angabe der zu traversierenden Objektklasse $TrootqObjSpec$ und der Zielobjektklasse $TleafObjSpec$ und einer Menge aus „Wegbeschreibungen“ $TpathSet$ zu den Traversierungstreffern.

*$TrootqObjSpec$:*

$TrootqObjSpec$ beschreibt die zu traversierende Objektklasse $Oclass$ und ihre Qualitätsstufe $OqLevel$, deren Eintreffen die Traversierung auslösen soll, sowie eine optionale Bedingung $TabstrBoolExpr$. Die Bedingung $TabstrBoolExpr$ wird entsprechend den Regeln aus Abschnitt 5.2 gebildet, wobei die freie Variable $O$ an die zu traversierende Objektklasse $TrootqObjSpec$ gebunden wird.

*$TleafObjSpec$:*

$TleafObjSpec$ beschreibt die Objektklasse der Zielobjekte. Die Qualitätsstufe der Zielobjekte muss nicht modelliert werden. Sie berechnet sich aus der Qualitätsstufe der Wurzelobjektklasse $TrootqObjSpec$.

*$TpathSet \rightarrow (ThitPolicy, Tpath) \mid (ThitPolicy, Tpath), TpathSet$:*

Mittels dieser Regel wird die Menge der Pfade vom Wurzelobjekt zu den verschiedenen Traversierungstreffern abgeleitet. $ThitPolicy$ legt das Verhalten bei Traversierungstreffern fest. Besitzt $ThitPolicy$ den Wert $tFirst$, so wird beim ersten Treffer auf diesem Pfad die Traversierung beendet. Beim Wert $tLast$ wird dagegen der letzte Treffer markiert.

*$Tpath$:*

Ein einzelner Pfad besteht aus einer Aneinanderreihung von Pfadsequenzen $TpathSequence$, wobei für jede Sequenz mittels $TtravPolicy$ angegeben wird, ob diese Sequenz einmal ($TtravPolicy = tNonRecursive$) oder sooft wie möglich ($TtravPolicy = tRecursive$) durchlaufen werden soll. Diese Unterscheidung ist notwendig, da im **WEP**-Modell auch rekursive Objektstrukturen möglich sind.

$TpathSectionSpec$:

$TpathSectionSpec$ legt eine Wegsektion fest. Sie beschreibt also, über welche Relation $Trelation$ zur nächsten Subobjektebene vorangegangen werden soll. Auch hier ist eine optionale Angabe einer Bedingung $TabstrBoolExpr$ möglich, die auf das Subobjekt angewandt wird, auf welche die $Trelation$ verweist (Bindung der freien Variablen $O$ an die Subobjektklasse).

*Beispiel 4: Ableitung eines auf dem Datenmodell von Abbildung 5.1 basierenden Traversierungsmerkmals, das alle noch nicht DMU-überprüften Elementarteile als Treffer markiert.*

Die Traversierung kann auf Objekte der Klasse *CRQclass* in der Qualitätsstufe *checked* angewandt werden. Als Traversierungstreffer kommen Objekte der Klasse *ElementaryPartClass* in Frage. Eine Analyse der Qualitätsstufen der Klasse *CRQclass* ergibt, dass alle Traversierungstreffer Objektversionen in der Qualitätsstufe *raw* sein werden (vergleiche Abschnitt 5.3 für die Definition der Qualitätsstufen für diese Beispielklasse).

$$
\begin{aligned}
&TraverseFeature \rightarrow (Tname, TrootqObjSpec, TleafObjSpec, \{TpathSet\}\\
&\quad \rightarrow^{*} (checked.TV1, (CRQClass, checked), (ElementaryPartClass), \{TpathSet\}\\
&\quad \rightarrow\ (checked.TV1, (CRQClass, checked), (ElementaryPartClass), \{Tpath, TpathSet\}\\
&\quad \rightarrow\ (checked.TV1, (CRQClass, checked), (ElementaryPartClass),\\
&\qquad \{(ThitPolicy, TpathSequence), TpathSet\}\\
&\quad \rightarrow^{*} (checked.TV1, (CRQClass, checked), (ElementaryPartClass),\\
&\qquad \{(tFirst, (TpathSectionSpec - TpathSequence)), TpathSet\}\\
&\quad \rightarrow (checked.TV1, (CRQClass, checked), (ElementaryPartClass),\\
&\qquad \{(tFirst, ((Trelation, TtravPolicy, TabstrBoolExpr) - TpathSequence))TpathSet\}\\
&\quad \rightarrow^{*} (checked.TV1, (CRQClass, checked), (ElementaryPartClass),\\
&\qquad \{(tFirst, ((consistsOfMods, tNonRecursive, SuperModuleClass.state \neq DMUchecked^{1})\\
&\qquad\quad -TpathSequence)), TpathSet\}\\
&\quad \rightarrow^{*} (checked.TV1, (CRQClass, checked), (ElementaryPartClass),\\
&\qquad \{(tFirst, ((consistsOfMods, tNonRecursive, SuperModuleClass.state \neq DMUchecked)\\
&\qquad -((consistsOfParts, tNonRecursive, ElementaryPartClass.state \neq DMUchecked)))),\\
&\qquad TpathSet\}\\
&\quad \rightarrow^{*} (checked.TV1, (CRQClass, checked), (ElementaryPartClass),\\
&\qquad \{(tFirst, ((consistsOfMods, tNonRecursive, SuperModuleClass.state \neq DMUchecked)\\
&\qquad\quad -((consistsOfParts, tNonRecursive, ElementaryPartClass.state \neq DMUchecked)))),\\
&\qquad\ (tFirst, ((consistsOfMods, tNonRecursive, SuperModuleClass.state \neq DMUchecked)\\
&\qquad\quad -((consistsOfSubMods, tRecursive, SuperModuleClass.state \neq DMUchecked)\\
&\qquad\quad -(consistsOfParts, tNonRecursive, ElementaryPartClass.state \neq DMUchecked))))))\}
\end{aligned}
$$

### 8.3.3 Abschließende Illustration der Traversierungskonzepte

Die soeben beschriebenen Konzepte der Traversierung sollen anhand eines konkreten Szenarios demonstriert werden, das sich an das Beispiel aus Teil I der Arbeit anlehnt. Der dort beschriebene Prozess (vergleiche Abbildung 2.2) soll ermöglichen:

- Die Aktivität *DefineChangeRequest* erzeugt einen ersten Änderungsauftrag (*CRQ*), indem sie eine Objektversion der Klasse *CRQClass* in der Qualitätsstufe *specified* erzeugt, der einen noch veränderbaren Änderungsumfang festlegt. Zur Vereinfachung des Beispiels spezifiziert ein Änderungsauftrag nur einzelne Bauteile und nicht noch zusätzlich Module. Das Datenmodell be-

[1] Dies ist die abkürzende Schreibweise für: $NOT\ AttrIsEq(SuperModuleClass, state, DMUchecked)$

steht dadurch einzig aus den zwei Klassen *CRQClass* mit den Qualitätsstufen *specified*, *rawParts*, *closed*, *finalParts* und *released* sowie der über die Relation *concistsOfParts* verbundenen Klasse *ElementaryPartClass* mit den Qualitätsstufen *specified*, *raw* und *final*.

- Dieser Änderungsauftrag wird als vorläufiges Datum weitergegeben. Es soll traversiert werden und alle Bauteile, die noch nicht in der Qualitätsstufe *final* und noch nicht DMU-geprüft sind, sollen Konstrukteuren über die Aktivität *DesignPart* angeboten werden.
- Änderungswünsche am Bauteileumfang sollen die Konstrukteure den Änderungsverantwortlichen mitteilen können.
- Die Konstrukteure sollen einen ersten vorläufigen Lösungsvorschlag ihres zu bearbeitenden Bauteils als *Rohgeometrie* (entspricht Qualitätsstufe *raw*) zu einer DMU-Prüfung anbieten.
- Auf Basis der Rohgeometrien wird eine erste DMU-Überprüfung durchgeführt. Diese Aufgabe obliegt der Verantwortung des Bearbeiters der Aktivität *CheckDMU*. Verbesserungswünsche bezüglich Änderungsumfang und Konstruktionsaufgaben sollen dem Verantwortlichen des Änderungsauftrags beziehungsweise den Bauteilkonstrukteuren mitgeteilt werden können. Neben einer vorläufigen DMU-Prüfung auf Basis von Rohgeometrien ist für die Freigabe des Änderungsauftrags eine abschließende Überprüfung in der Qualitätsstufe *finalParts* notwendig.
- Liegen alle Bauteile eines Änderungsauftrags DMU-geprüft in der Qualitätsstufe *raw* vor, so wird dieser „geschlossen“ und der Bearbeitungsschritt *DefineChangeRequest* beendet. Ein Verändern dieses Auftrags ist dann nur noch durch das Erzeugen eines neuen Änderungsauftrags möglich. Dies soll mittels einer Iterationsschleife realisiert werden.

**Umsetzung des Szenarios im WEP-Modell**

Ausgangspunkt der Modellierung ist die Aktivität *DefineChangeRequest*, die als Ausgabe einen Änderungsauftrag (*CRQ-Objekt*) der Klasse *CRQClass* in den Qualitätsstufen *specified* und *closed* erzeugt. Beide Qualitätsstufen werden dem einzigen Returncode *ok* zugewiesen, wobei die Qualitätsstufe *closed* das Endergebnis und die Qualitätsstufe *specified* ein notwendiges Zwischenergebnis repräsentiert. Zur Darstellung der Iterationsschleife erhält diese Aktivität noch einen optionalen Eingabeparameter, der die „alte“ Version des Änderungsauftrags (Qualitätsstufe *closed*) enthält, die als Grundlage bei der Erzeugung des neuen Änderungsauftrags dienen kann.

Da die Zahl der zu ändernden Bauteile variabel ist, wird eine dynamische Parallelität zur Modellierung dieses Sachverhalts benötigt. Die Traversierung wird angestoßen, wenn ein *CRQ*-Objekt in den Qualitätsstufen *specified* beziehungsweise *closed* vorliegt. Beide Qualitätsstufen werden durch die Aktivität *DefineChangeRequest* bereitgestellt. Für die Modellierung der Traversierung ist außerdem das gewünschte Ergebnis am Ende der dynamischen Parallelität festzulegen. Im betrachteten Szenario wird für die Aktivität *DMU-Check* ein *CRQ*-Objekt in den Qualitätsstufen *rawParts* und *finalParts* (Alle Bauteile liegen in der Qualitätsstufe *raw* beziehungsweise *final* vor.) benötigt. Mit dem Abschluss der dynamischen Parallelität muss also ein Änderungsauftrag in diesen beiden Qualitätsstufen geliefert werden. In Abbildung 8.5 wird dieser Modellierungsstand dokumentiert.

Objektversionen der Klasse *ElementaryPartClass*, die noch nicht in der Qualitätsstufe *final* vorliegen und noch nicht DMU-geprüft sind, werden als Traversierungstreffer den Bauteilkonstrukteuren bereitgestellt. Abhängig von der Qualitätsstufe der traversierten *CRQ*-Objektversion werden die Blöcke

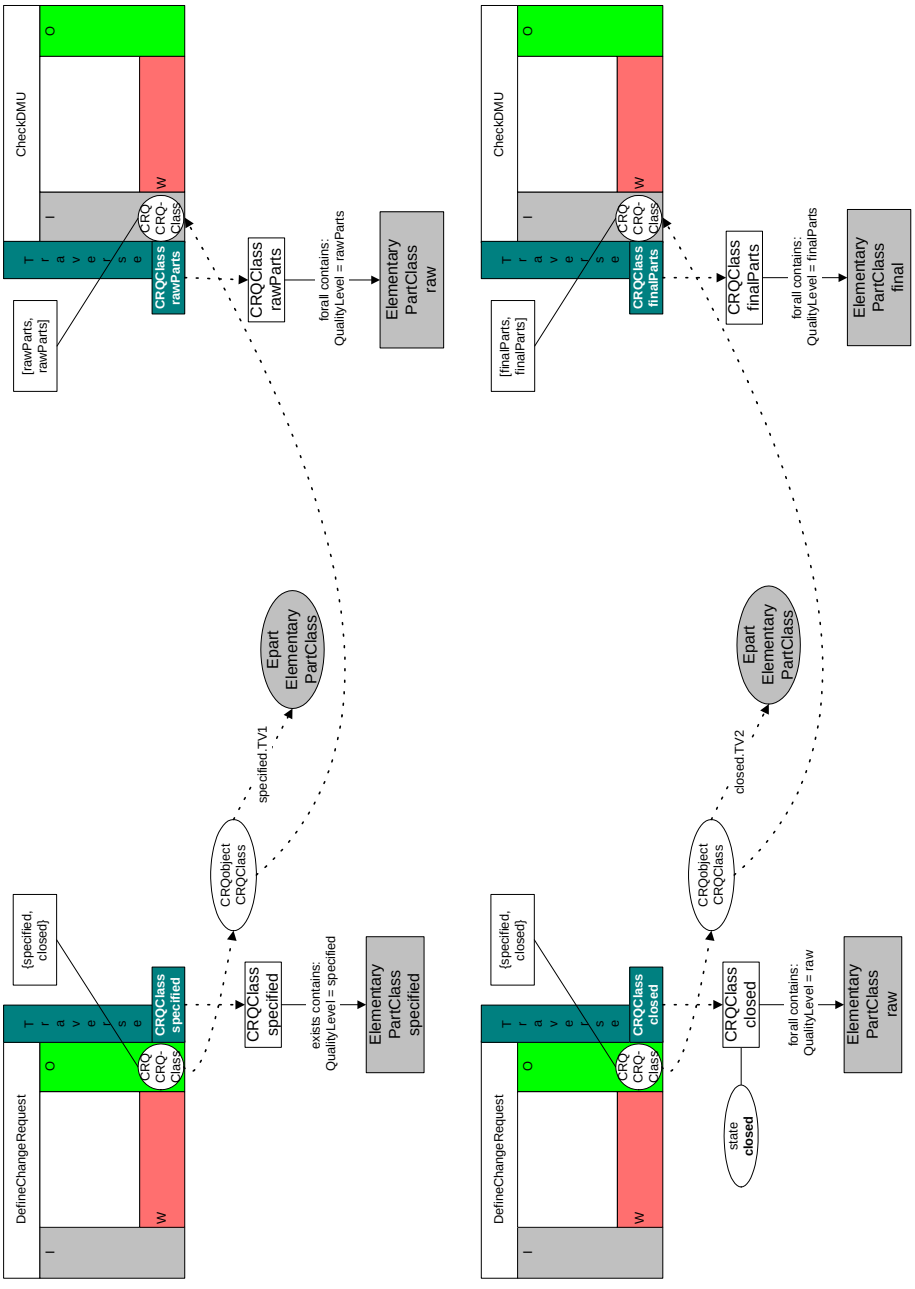



**Abbildung 8.5:** Analyse der Start- und Zielqualitätsstufen der Klasse CRQClass

innerhalb der dynamischen Parallelität als Eingabeparameter eine Objektversion der *ElementaryPart-Class*-Klasse in den Qualitätsstufen *specified* beziehungsweise *raw* erhalten. Die Blöcke werden eine Objektversion derselben Klasse in den Qualitätsstufen *raw* beziehungsweise *final* als Ausgabe liefern. Abbildung 8.6 zeigt diesen Modellierungsstand.

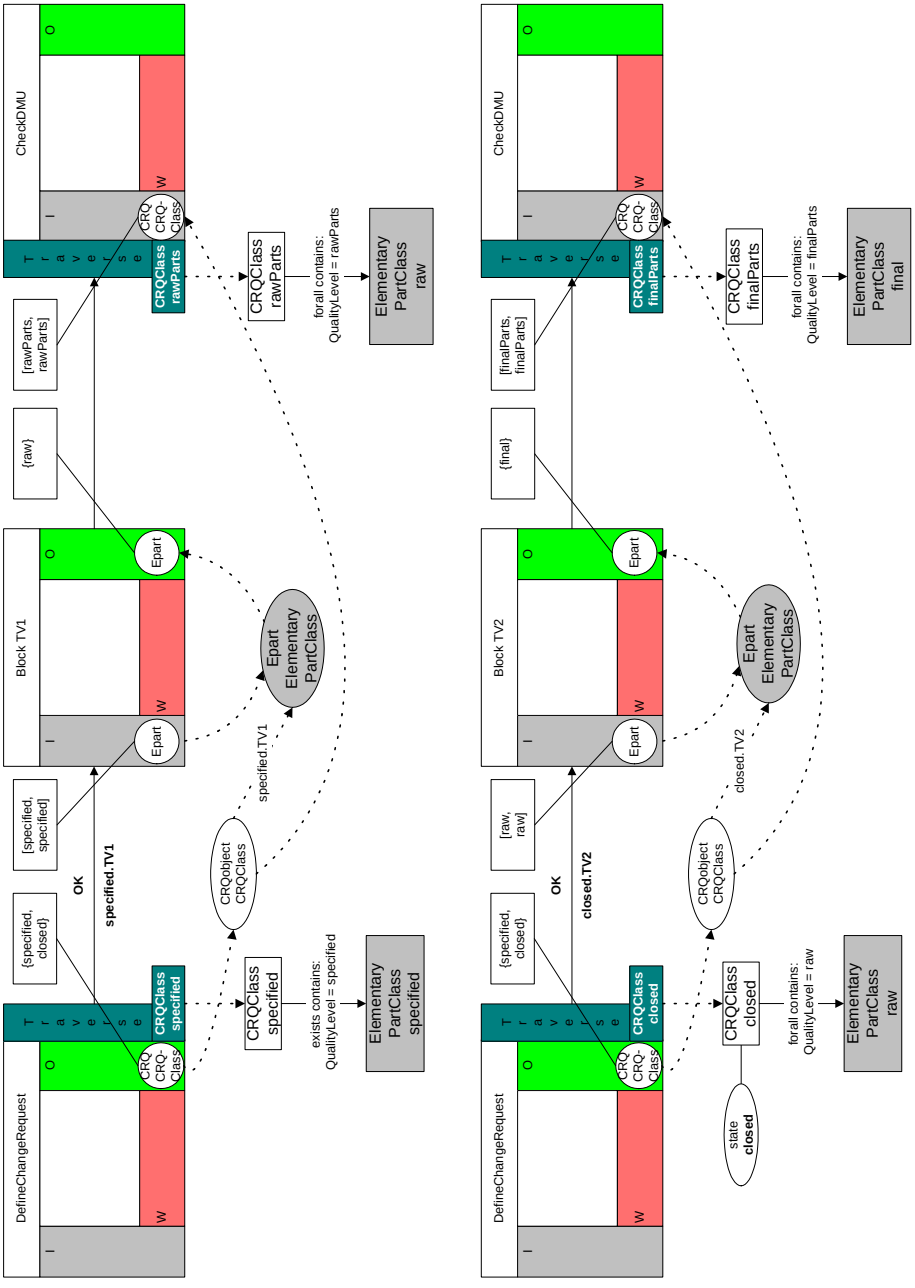



**Abbildung 8.6:** Definition der Blöcke innerhalb der dynamischen Parallelität

Die zwei in Abbildung 8.6 gezeigten Blöcke können zu einer Aktivität für beide Traversierungsmerkmale mit Erweiterung der Qualitätsstufen für Ein- und Ausgabeparameter zusammengefasst werden, da beide Blöcke die *DesignPart*-Aktivität repräsentieren. Erweitert um die Iterationsschleife ergibt sich das in Abbildung 8.7 gezeigte Workflow-Modell für das skizzierte Szenario.

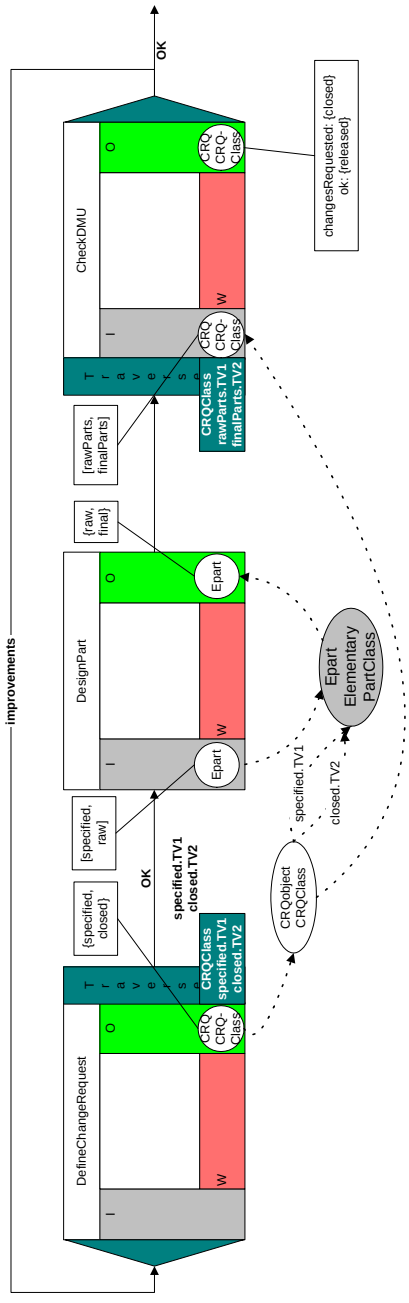



**Abbildung 8.7: WEP**-Workflow für das skizzierte Änderungsprozessszenario

Sobald die Aktivität *DefineChangeRequest* eine *CRQ*-Objektversion in der Qualitätsstufe *specified* weitergegeben hat, wird das Traversierungsmerkmal *specified.TV1* angestoßen. Für jeden Traversierungstreffer wird eine Aktivität $DesignPart_i$ und ein globales Objekt $Epart_i$ erzeugt, ein Traversierungstreffer zugewiesen und den entsprechenden Konstrukteuren angeboten. Wurde von jeder $DesignPart_i$-Aktivität eine Objektversion in der Qualitätsstufe *raw* weitergegeben, so ist die Bedingung am Traversierungsende erfüllt, es wird eine neue Objektversion des Änderungsauftrags erzeugt und die Bauteile der Qualitätsstufe *raw* werden hinzugefügt. Basierend auf diesem Änderungsauftrag kann die Aktivität *CheckDMU* eine erste DMU-Überprüfung durchführen. Die drei Aktivitäten können damit zeitlich überlappend in einer Simultaneous-Engineering-Phase bearbeitet werden. Die Aktivität *DefineChangeRequest* kann erst nach Weitergabe einer CRQ-Objektversion in der Qualitätsstufe *closed* beendet werden. Diese Qualitätsstufe muss auch erreicht sein, um die Iterationsschleife aktivieren zu können.

Durch das **WEP**-Konstrukt dynamische Parallelität konnte der skizzierte Änderungsprozess einfach und kompakt modelliert werden. Modellierungsversuche mit anderen Workflow-Management-Systemen scheiterten dagegen [BKJ+99].

# Kapitel 9

# Datenflussanalyse

Primäres Ziel einer Datenflussanalyse ist es, zur Modellierungszeit zu überprüfen, ob zur Laufzeit alle Prozessschritte eines Workflows mit Eingabedaten versorgt sind.

Die Datenflussanalyse hat im **WEP**-Modell darüber hinaus noch weitere Anforderungen zu erfüllen. Sie muss nicht nur das Bereitstellen von Daten überprüfen, sondern dabei auch noch analysieren, ob die bereitgestellten Daten in den richtigen Qualitätsstufen vorhanden sind. Das Minimalziel ist hierbei, dass die Qualitätsstufen eines Aktivitätenausgabeobjekts, die ein Endergebnis repräsentieren, innerhalb des Qualitätsbereichs des lesenden Eingabeparameters liegen. Damit ist die Datenkonsistenz sichergestellt. Andernfalls sind Inkonsistenzen möglich, wie aus dem Beispiel der Abbildung 9.1 ersichtlich wird.

Zur Verkürzung der Prozessdurchlaufzeiten ist es jedoch erstrebenswert, dass auch erzeugte Zwischenergebnisse verarbeitet werden, damit eine vorzeitige Datenweitergabe auch zu einer möglichst hohen zeitlichen Überlappung sequentieller Prozessschritte führt. Die Datenflussanalyse sollte deshalb das Datenflussmodell daraufhin überprüfen, um den Modellierer auf keine oder geringe Überlappungen hinweisen zu können.

Zuletzt muss bei der Datenflussanalyse berücksichtigt werden, dass die bereitgestellten Ausgabeobjekte vom gewählten Returncode abhängen.

## 9.1 Korrekte, minimale und SE-optimierte Datenflüsse

*Definition 2: Korrekte, minimale und SE-optimierte Datenflussmodelle*

Sei $dfmodel$ das Datenflussmodell eines beliebigen Workflows.

- Das Datenflussmodell heißt *korrekt*, wenn für jeden Prozessschritt eines Workflows beim Start für alle notwendigen Eingabeparameter *endgültige* Objektversionen in *einer* der geforderten Qualitätsstufen vorliegen, unabhängig davon, welcher Kontrollflusspfad durch den Workflow-Graphen zur Laufzeit gewählt wird.
- Ein korrektes Datenflussmodell heißt *minimal*, wenn von allen globalen Objekten mindestens eine Objektversion gelesen werden kann.

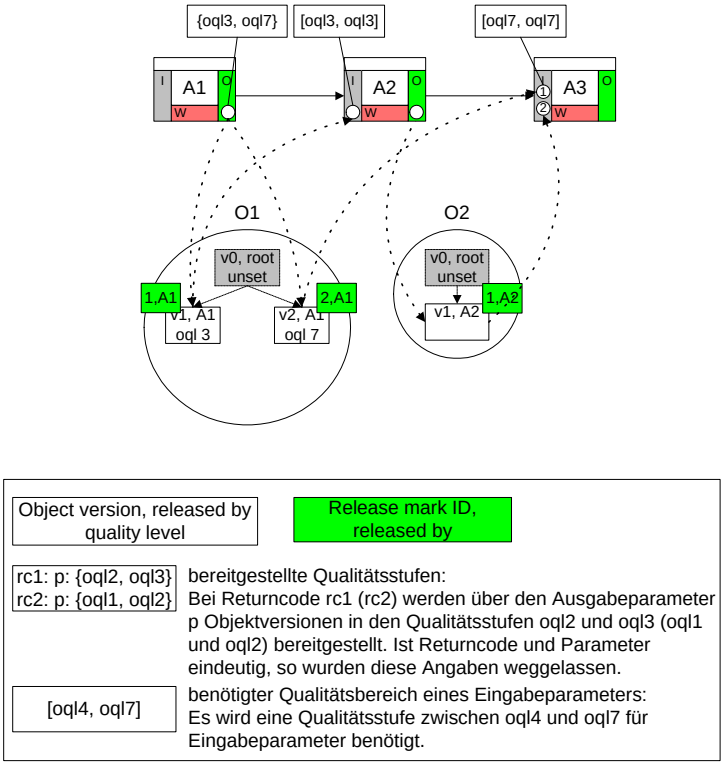



**Abbildung 9.1:** Gefahr von Dateninkonsistenzen aufgrund unterschiedlicher Release-Stände: Parameter $1$ von $A3$ liest von Objekt $O1$ die Objektversion $v2$ in Qualitätsstufe $oql_7$, Parameter $2$ liest indirekt die Qualitätsstufe $oql_3$ (Objektversion $v1$) des Objekts $O1$, da auf dieser Basis $A2$ eine Objektversion vom Objekt $O2$ für $A3$ erzeugt hat.

- Ein minimales Datenflussmodell heißt *SE-optimiert*[1], wenn für jede in einem globalen Objekt abgelegte Qualitätsstufe eine diese Qualitätsstufe beinhaltende Objektversion gelesen wird.

  Wird eine Objektversion in einer bestimmten Qualitätsstufe erzeugt, so existiert also immer mindestens eine Aktivität, die eine Objektversion dieser Qualitätsstufe auch liest.

Ein korrektes Datenflussmodell verhindert ein Blockieren eines Workflows zur Laufzeit aufgrund fehlender Eingabedaten. Ein minimales Datenflussmodell stellt darüber hinaus noch klar, dass jeder Ausgabeparameter auch von einer nachfolgenden oder parallelen Aktivität benötigt wird. Ein SE-optimiertes Datenflussmodell garantiert eine möglichst hohe zeitliche Überlappung sequentieller Aktivitäten, da auch jedes Zwischenergebnis von nachfolgenden oder parallelen Aktivitäten benötigt wird.

Die Analyse, ob ein modelliertes Datenflussmodell den Korrektheitsbegriffen genügt, muss spätestens nach Abschluss der Datenflussmodellierung erfolgen. Erstrebenswert ist es jedoch, dass Datenflussmodellierer bereits während des Aufbaus eines Datenflussmodells auf mögliche Konflikte hingewiesen werden. Dazu ist es notwendig, dass die Analyse auch auf noch unvollständige Datenflussmodelle angewandt werden kann.

[1] SE steht für Simultaneous-Engineering.

Da die Datenflussanalyse vor der Workflow-Ausführung zu erfolgen hat, müssen mindestens hinreichende Kriterien entwickelt werden, die, wenn sie ein modellierter Workflow einhält, sicherstellen, dass dieser Workflow auch die in Definition 2 aufgestellten Laufzeitanforderungen erfüllt. Im Folgenden werden solche zur Modellierungszeit überprüfbare Kriterien festgelegt und es wird ein formales Datenflussanalyseverfahren zu deren Einhaltung hergeleitet.

## 9.2 Modellierungsregeln für Datenflüsse

Dieser Abschnitt spezifiziert allgemein und anschaulich Kriterien, die Datenflussmodelle im **WEP**-Modell einhalten müssen:

1. *Eindeutigkeit* und *Vollständigkeit:* Von jedem Ausgabeparameter einer Aktivität muss *genau eine* Datenflussausgangskante ausgehen und zu jedem *notwendigen* Eingabeparameter einer Aktivität muss *genau eine* Datenflusseingangskante hinführen.

2. *Datenversorgung aus parallelen Zweigen:* Datenflussabhängigkeiten zwischen parallelen Zweigen sind erlaubt, da unabhängig vom gewählten Kontrollflusspfad die Datenversorgung gewährleistet ist (vergleiche Abbildung 9.2). Die Synchronisation der Aktivitäten aus unterschiedlichen parallelen Zweigen erfolgt dynamisch zur Laufzeit.

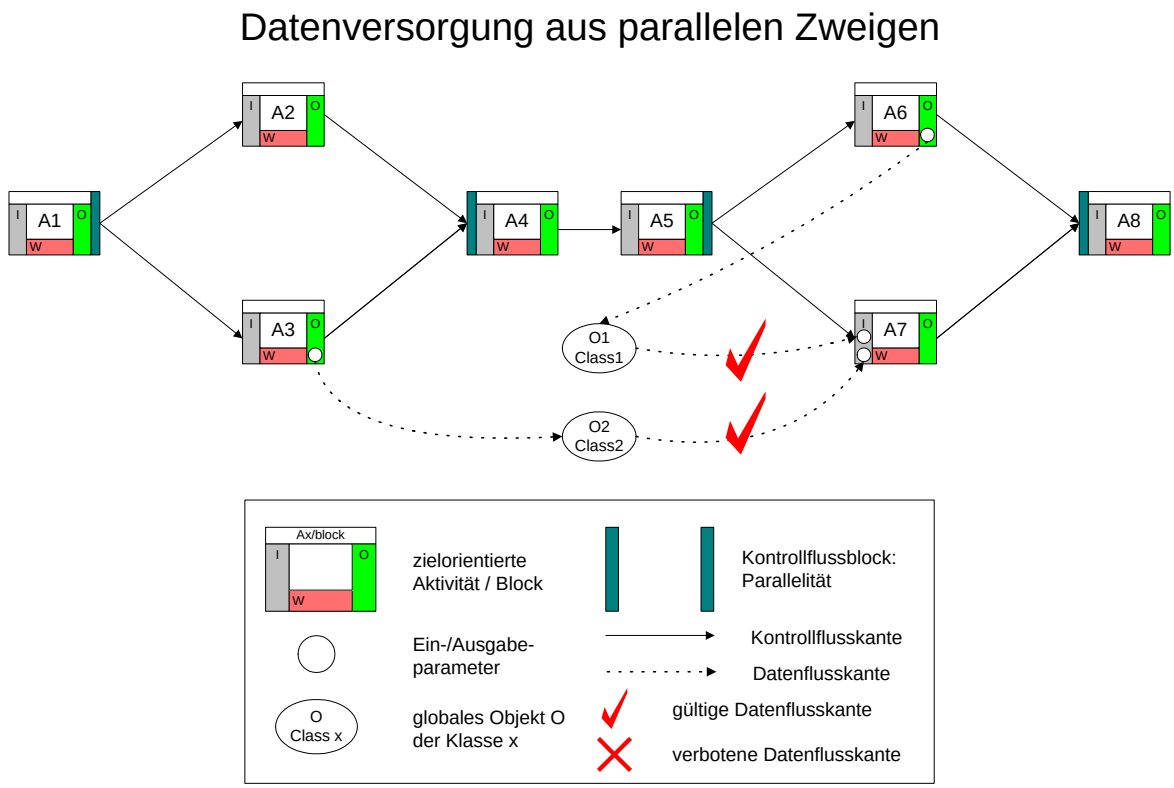



**Abbildung 9.2:** Datenflussregeln bei Parallelität

3. *Datenversorgung aus bedingten Zweigen und bei Schleifen:* Dagegen sind Datenflusskanten zwischen bedingten Zweigen illegal, da immer Kontrollflusspfade existieren, bei denen keine Datenversorgung sichergestellt werden kann (vergleiche Abbildung 9.3).

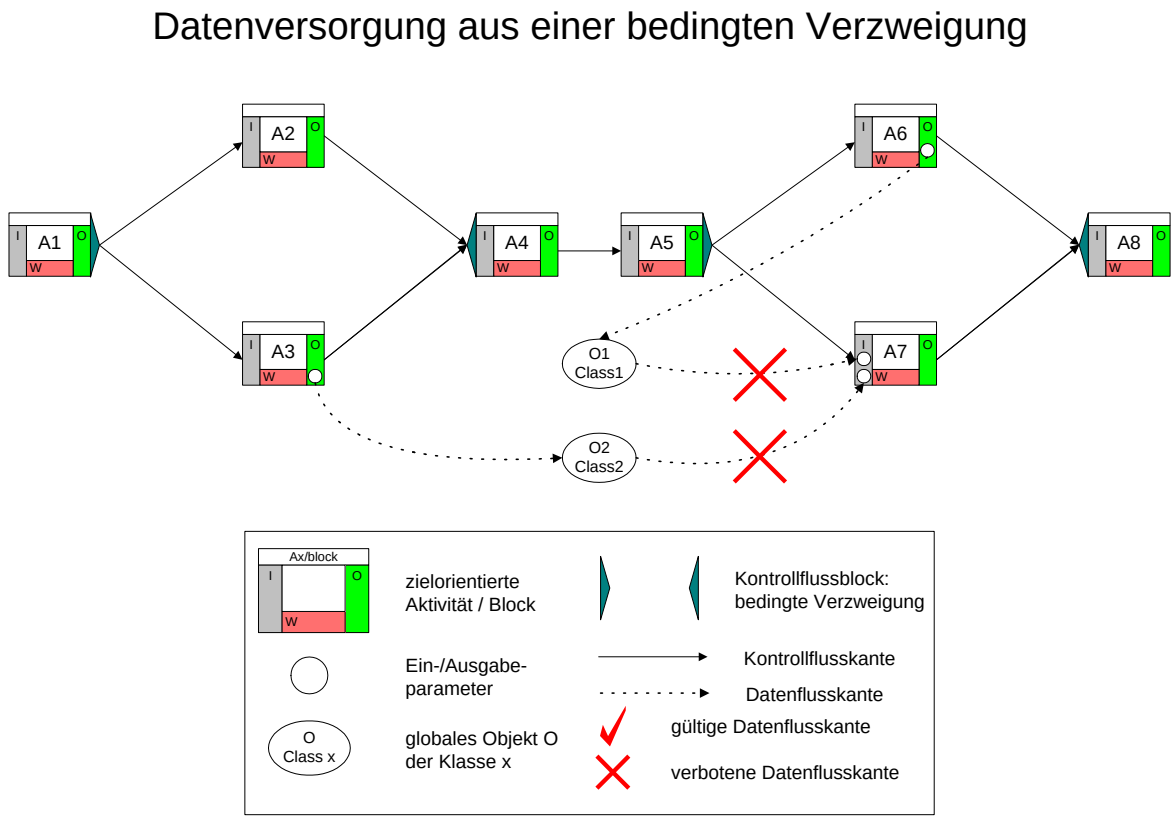



**Abbildung 9.3:** Datenflussregeln bei bedingten Verzweigungen

Aus diesem Grund unterliegen auch Datenflüsse bei Schleifen denselben Restriktionen (vergleiche Abbildung 9.4).

4. *Klassenkonformität:* Es darf nur dann eine Datenflusskante zwischen Aktivitätenparameter und globalem Objekt gezogen werden, wenn bei beiden die gleiche Objektklasse spezifiziert wurde.

5. *Kontrollflusskonformität:* Die Datenflussrichtung darf nicht der Kontrollflussrichtung widersprechen (*data flow follows control flow*). Damit darf eine Aktivität $A$ über Datenflusseingangskanten (und globale Objekte) nur mit Aktivitäten verbunden sein, die gemäß Kontrollfluss entweder Vorgängeraktivitäten sind oder auf zu $A$ parallelen Zweigen liegen.

6. *Qualitätsstufenüberschneidung:* Unabhängig vom zur Laufzeit gewählten Kontrollflusspfad müssen die durch einen Aktivitätenausgabeparameter bereitgestellten Qualitätsstufen innerhalb des Qualitätsbereichs des über Datenflusskanten verbundenen Eingabeparameters einer im Kontrollfluss nachfolgenden oder parallelen Aktivität liegen. (vergleiche Abbildung 9.5). Diese Regel lässt sich daraufhin abschwächen, dass nur die Qualitätsstufen, die Endergebnisse von Aktivitäten repräsentieren, innerhalb des Qualitätsbereichs der lesenden Aktivität liegen müssen. Weitergegebene Zwischenergebnisse (niedrigere Qualitätsstufen) werden dann even-

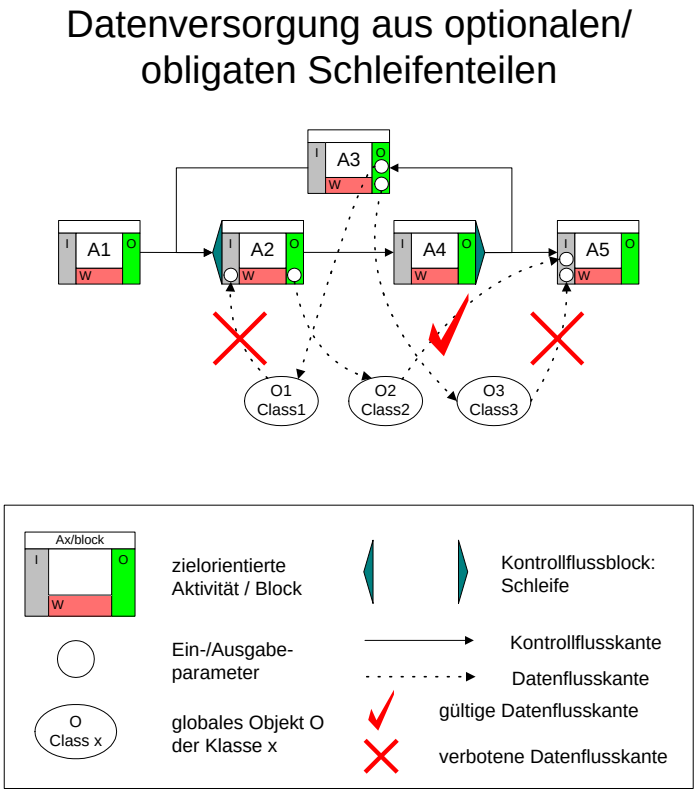



**Abbildung 9.4:** Datenflussregeln bei Schleifen

tuell unnötigerweise durch die schreibende Aktivität bereitgestellt. Bei der Datenflussanalyse ist es deshalb sinnvoll, zwischen End- und Zwischenergebnis repräsentierenden Qualitätsstufen zu unterschieden. Ein SE-optimiertes Datenflussmodell muss auch alle Zwischenergebnisse repräsentierenden Qualitätsstufen verarbeiten.

## 9.3 Formales Datenflussanalyseverfahren

Wie kann nun die Analyse der Datenflüsse im **WEP**-Modell gestaltet werden, um die definierten Korrektheitsbegriffe einzuhalten?

Während die Überprüfung der Minimalität eines Datenflussmodells trivial ist, gestaltet sich die Analyse von Korrektheit und SE-Optimalität schwieriger: Eine intuitive Vorgehensweise besteht darin, zuerst alle möglichen Kontrollflusspfade vom Start des Workflows bis zu der in der Datenflussanalyse aktuell betrachteten Aktivität zu bestimmen, für jeden dieser Kontrollflusspfade die über diesen Pfad beschriebenen globalen Objekte zu berechnen und danach zu analysieren, welche der gefundenen globalen Objekte auf *allen* Pfaden beschrieben werden. Sind die so gefundenen globalen Objekte klassenkonform und „passen“ die bereitgestellten Qualitätsstufen, so können diese globalen Objekte als Ausgangspunkt für Datenflusseingangskanten zur betrachteten Aktivität dienen, da sie unabhängig vom zur Laufzeit gewählten Kontrollflusspfad immer mit geeigneten Objektversionen belegt sein werden. Damit ist das Datenflussmodell sicherlich korrekt beziehungsweise SE-optimiert.

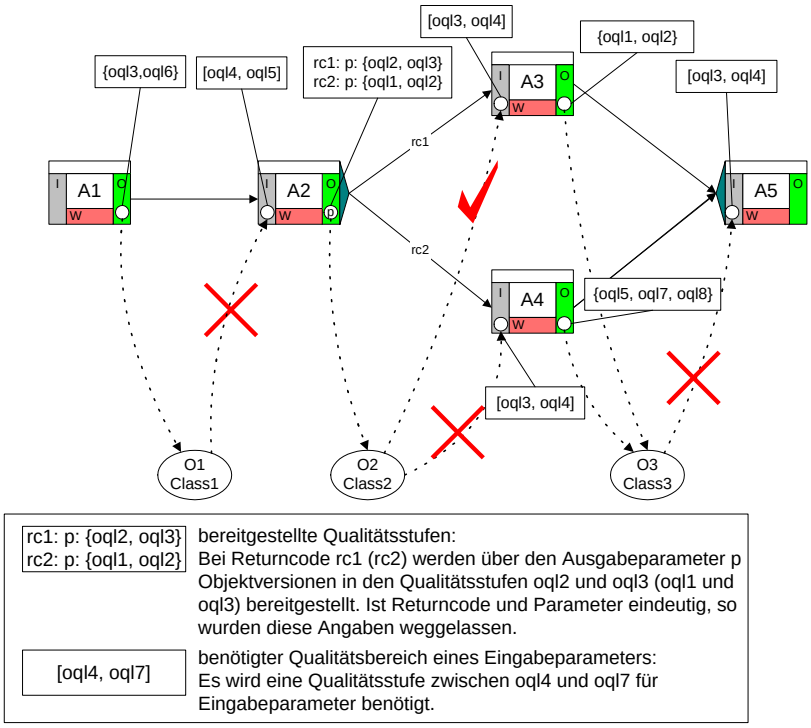



**Abbildung 9.5:** Fehlende Überschneidungen bei den Qualitätsbereichen führen zum Blockieren eines Workflows und bergen die Gefahr von Dateninkonsistenzen.

Eine solche Vorgehensweise führt jedoch zu vielen Fallunterscheidungen[2] und unterstützt nicht die angestrebte schrittweise Analyse auch unvollständiger Datenflussmodelle.

Im **WEP**-Modell wird deshalb ein anderer, *blockorientierter*, Weg eingeschlagen. Statt die verschiedenen Kontrollflusspfade getrennt zu untersuchen, wird *ein kumulierter* Kontrollflusspfad bestimmt. Darunter versteht man eine *Sequenz* von Aktivitäten von einer *Startaktivität* $A_s$ zu einer *Endaktivität* $A_e$, bei der für jede Aktivität $A_i$ zwischen $A_s$ und $A_e$ gilt: Sie ist entweder eine elementare Aktivität des Workflows oder eine *Blockaktivität*, die alle möglichen Wege durch einen Block zu einer Aktivität zusammenfasst (*kumuliert*). Eine Blockaktivität repräsentiert damit aus Sicht der Datenflussanalyse einen Block des Workflows als Ganzes. Sie ist so definiert, dass sie die gleichen Ausgaben wie die Aktivitäten des Blocks produziert, unabhängig davon, welcher Kontrollflusspfad innerhalb des Blocks später zur Laufzeit durchlaufen wird.

Durch diesen Ansatz kann jeder Block einzeln analysiert werden, da die Analyse eines beliebig komplexen Workflow-Graphen auf eine schrittweise Analyse elementarer Kontrollflusskonstrukte zurückgeführt werden kann. Bei Veränderungen des Datenflussmodells müssen auch immer nur die Blöcke erneut analysiert werden, bei denen Datenflussveränderungen erfolgt sind. Diese Vorgehensweise erleichtert die angestrebte Analyse noch unvollständiger Datenflussmodelle.

Neben der Analyse der Vorgängerblöcke einer Aktivität $A$ müssen auch noch die zu ihr parallelen Kontrollflusspfade untersucht werden, da in den anvisierten Anwendungsgebieten die Modellierung

[2] Beispielsweise muss differenziert werden, ob sich zwei Kontrollflusspfade aufgrund einer bedingten oder parallelen Verzweigung unterscheiden.

von Datenabhängigkeiten zwischen parallelen Prozessschritten möglich sein muss. Die Vorgehensweise bei der Datenflussanalyse wird exemplarisch in den Abbildungen 9.6 und 9.7 gezeigt.

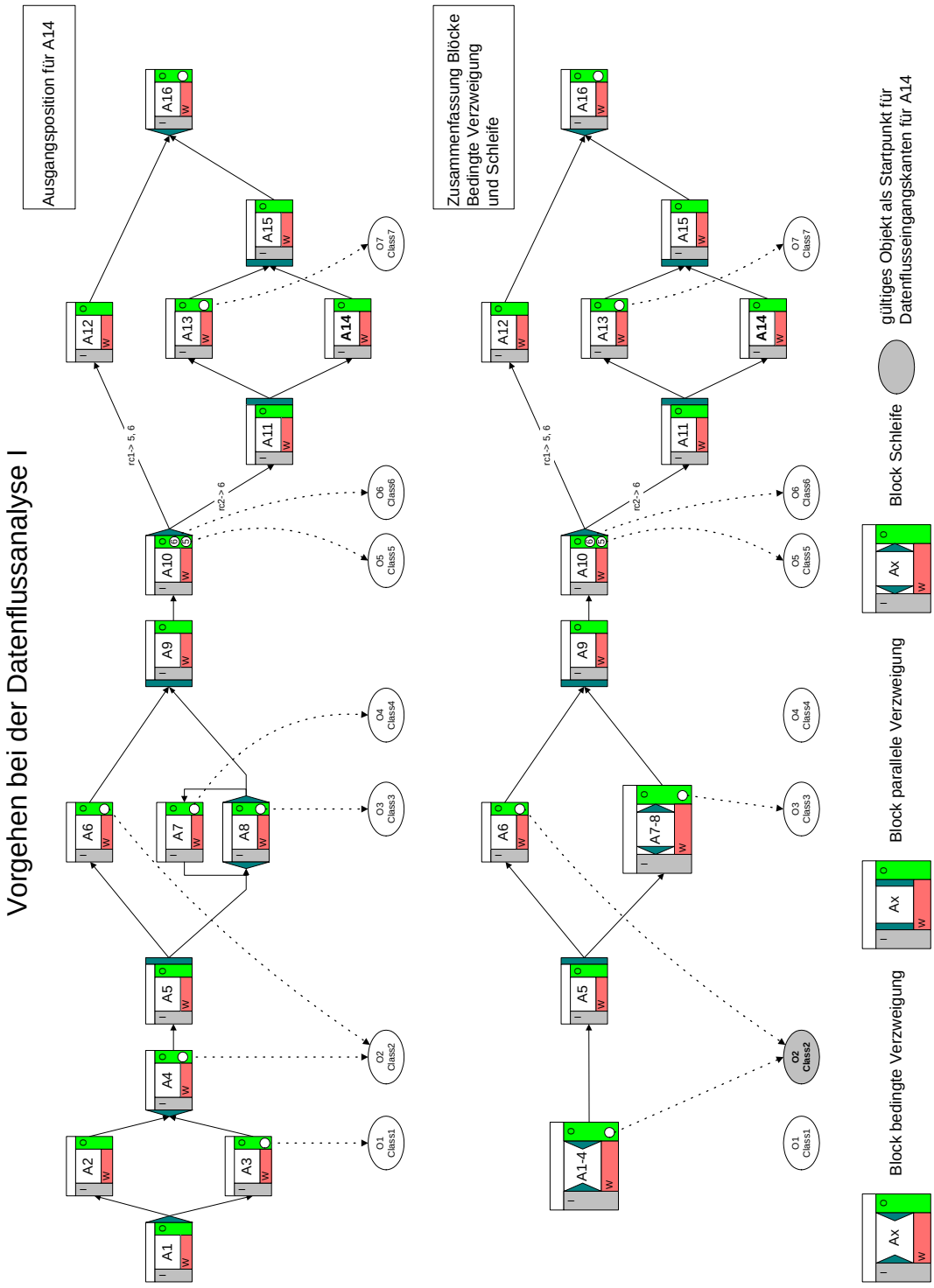



**Abbildung 9.6:** Vorgehen bei der Datenflussanalyse, Analyse der Vorgängerblöcke

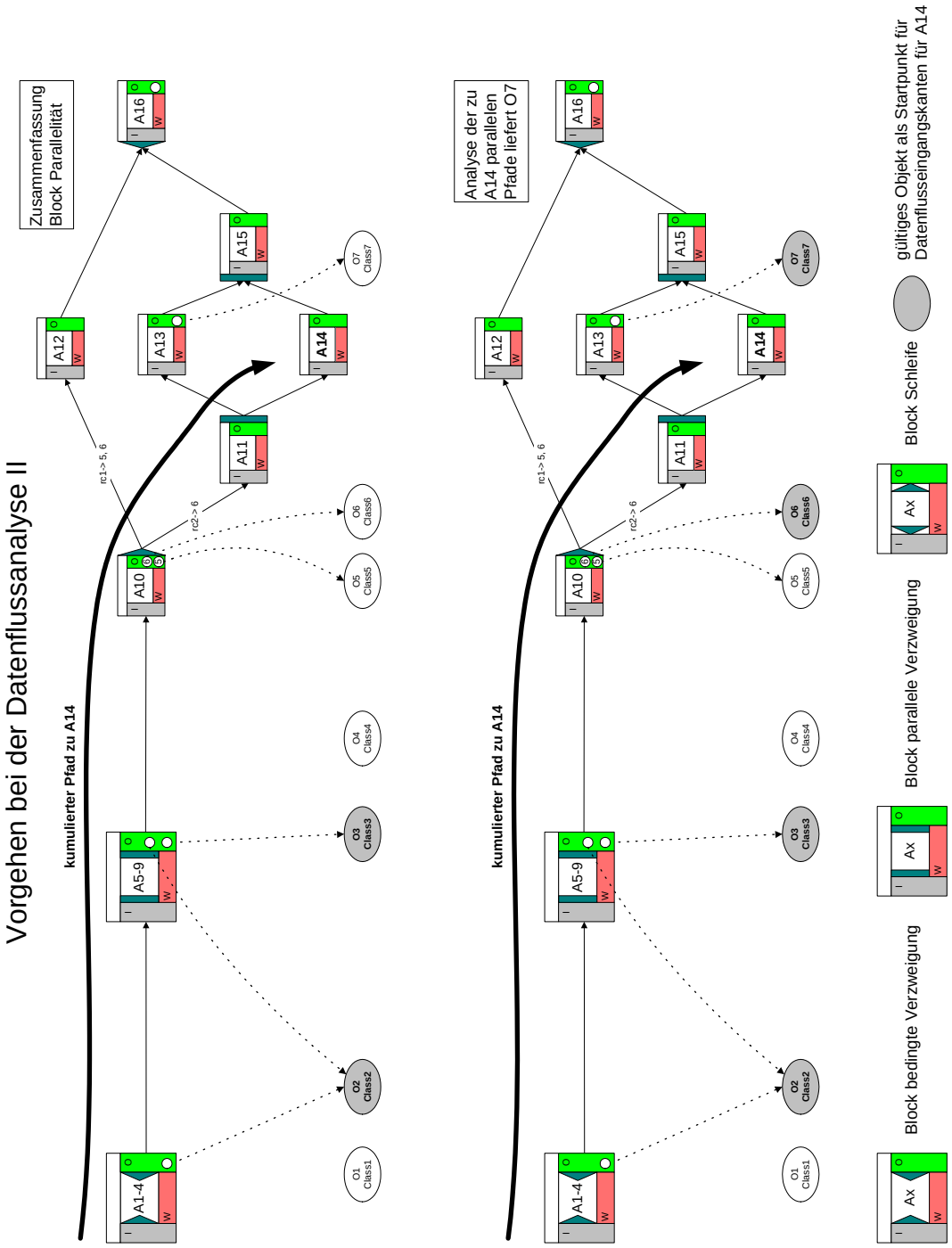



**Abbildung 9.7:** Vorgehen bei der Datenflussanalyse, Analyse Vorgängerblock und eigener Block. Bei Returncode rc2 muss nur Parameter 6 belegt werden, der in Objekt O6 abgelegt wird. Deshalb ist auch nur Objekt O6 und nicht auch O5 ein gültiges Objekt.

Die gefundenen globalen Objekte dienen als Ausgangspunkt zur Überprüfung der Klassenkonformität und der Qualitätsstufenüberschneidungen. Klassenkonformität vorausgesetzt ist das Datenflussmodell genau dann korrekt, wenn alle Datenflusseingangskanten, die zu notwendigen Eingabeparametern führen, ihren Ursprung bei globalen Objekten haben, die von kumulierten Vorgängeraktivitäten oder parallelen Aktivitäten mit Objektversionen belegt werden, deren Qualitätsstufen innerhalb des Qualitätsbereichs der Eingabeparameter der jeweiligen Aktivität liegen. Die Überprüfung der bereitgestellten Qualitätsstufen kann durch Analyse der Meilensteine der Aktivitäten erfolgen.

Werden bei der Analyse nur die Meilensteine berücksichtigt, die Endergebnisse repräsentieren,[3] so ist bei einer positiven Überprüfung das Datenflussmodell korrekt.

Werden zusätzlich noch die Zwischenergebnisse repräsentierenden Meilensteine berücksichtigt und liegen die zugeordneten Qualitätsstufen ebenfalls innerhalb des Qualitätsbereichs der Aktivitäteneingangsparameter, so ist das korrekte Datenflussmodell auch noch SE-optimiert.

Es wird im Folgenden gezeigt, wie der kumulierte Kontrollflusspfad formal hergeleitet werden kann. Die Herleitung erfolgt im Wesentlichen in drei Schritten. Zuerst wird dargestellt, wie sich aus der Workflow-Schemabeschreibung ableiten lässt, welche globalen Objekte mit welchen Qualitätsstufen durch einen beliebigen Kontrollflusspfad belegt werden (Abschnitt 9.3.2). Diese Vorgehensweise wird dann für jeden elementaren Kontrollflussblock des **WEP**-Modells angewandt. Mittels geeigneter Schnitt- und Vereinigungsmengenbildung werden über dessen Kontrollflusspfade die globalen Objekte samt Qualitätsstufen bestimmt, die durch diesen Block als Ganzes beschrieben werden (Abschnitt 9.3.3). Damit sind alle Grundlagen für den formalen Aufbau des kumulierten Kontrollflusspfads einer Aktivität geschaffen (Abschnitt 9.3.4). Abschnitt 9.3.5 zeigt noch die Herleitung der globalen Objekte inklusive Qualitätsstufen der dort zur Laufzeit gespeicherten Objektversionen, die durch Aktivitäten beschrieben werden, die auf bezüglich der betrachteten Aktivität parallelen Zweigen liegen. Mittels dieser Herleitungen können dann in Abschnitt 9.3.6 zur Modellierungszeit überprüfbare Kriterien definiert werden, die gemäß Definition 2 korrekte beziehungsweise SE-optimierte Datenflussmodelle sicherstellen.

### 9.3.1 Vorbereitende Definitionen

Im Gegensatz zur Beschreibung der **WEP**-Modellierungssprache, bei der eine *grammatikbasierte* Darstellung vorteilhafter ist, ist bei der Analyse und später auch für die Ausführung von **WEP**-Workflows eine *graphbasierte* Darstellung geeigneter, die nun schrittweise eingeführt wird. Basis der graphbasierten Darstellung ist eine *erweiterte* Aktivitätenmenge, auf der eine *Kontrollflussnachfolgerrelation* definiert wird, die wiederum zur Definition von *Kontrollflusskanten* eingesetzt wird:

*Definition 3: Kontrollflussknotenmenge* $ACTIVITIES^{ext}$

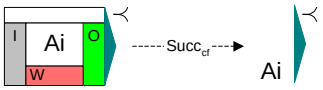
Sei $WF \in$ **WEP**-$WORKFLOWS$ ein beliebiger **WEP**-Workflow. Die Menge

$$ACTIVITIES^{ext} := ACTIVITIES(WF) \cup \{ {}_i\blacktriangleright^{\prec}, {}^{\succ}\blacktriangleleft_i, {}_i\blacktriangleright^{\circlearrowleft}, {}^{\circlearrowleft}\blacktriangleleft_i, {}_i\dashv^{st}, {}^{st}\vdash_i, {}_i\dashv^{tv}, {}^{tv}\vdash_i \}$$

enthält alle Knoten eines **WEP**-Workflows $WF$. Dies sind neben den Aktivitäten die Terminale für die Kontrollflusskonstrukte Verzweigung, Schleife, statische und dynamische Parallelität. Der Index $i$

[3] höchster Meilenstein eines Returncodes (siehe Abschnitt 6.2.2.3)

bezeichnet den Namen der mit dem jeweiligen Konstrukt assoziierten Aktivität. Die Kontrollflusskonstrukte repräsentierenden Knoten werden im weiteren Verlauf der Arbeit als *Kontrollflussknoten* bezeichnet. Zur Veranschaulichung sei die bei der Aktivität $Ai$ beginnende bedingte Verzweigung gezeigt:




Ist es nicht notwendig zwischen statischer und dynamischer Parallelität beziehungsweise zwischen bedingter Verzweigung und Schleife zu unterscheiden, so werden die Terminale $_i$▌ und ▐$_i$ beziehungsweise $_i$▶ und ◀$_i$ verwendet.

Basierend auf dieser Menge soll eine Nachfolgerrelation definiert werden, die gemäss Kontrollfluss die Menge aller zu einer Aktivität $A$ nachfolgenden Aktivitäten liefert:

*Definition 4: Nachfolgerrelation* $Succ_{cf}$

$$\begin{aligned} Succ_{cf} \quad &: \quad ACTIVITIES^{ext} \times \wp\,(ACTIVITIES^{ext})^{4} \\ N_i.Succ_{cf} \quad &:= \quad \{N_j : N_i \text{ ist im erweiterten Workflow-Graphen mit } N_j \\ &\qquad\quad \text{direkt über eine Kontrollflusskante verbunden,} \\ &\qquad\quad \text{die bei } N_i \text{ startet und bei } N_j \text{ endet.}\} \end{aligned}$$

Diese informelle Definition der Nachfolgerrelation $Succ_{cf}$ soll an dieser Stelle genügen. Die Abbildung 9.8 zeigt die Definition der Nachfolgerrelation in einer anschaulichen graphischen Darstellung. Die formale Definition der Nachfolgerrelation basiert auf Regeln über der Kontrollflussgrammatik aus Abschnitt 7.3. Sie ist, um alle Fälle abzudecken, sehr umfangreich und deshalb im Anhang A zu finden. In analoger Weise wird die Vorgängerrelation $Pred_{cf}$ definiert. $Succ^{*}_{cf}$ und $Pred^{*}_{cf}$ bezeichnen die transitive Hülle von $Succ_{cf}$ beziehungsweise $Pred_{cf}$.

Mittels Vorgänger- und Nachfolgerrelationen lassen sich Kontrollflusskanten spezifizieren, die ebenfalls auf der erweiterten Aktivitätenmenge basieren. Die Kontrollflusskonstrukte müssen von ihren assoziierten Aktivitäten getrennt und mit ihnen über Kontrollflusskanten verknüpft werden. Diese neuen Kontrollflusskanten erhalten dabei die identischen Returncodebeschriftungen wie die bereits vorhandenen ein- und ausgehenden Kontrollflusskanten. Die jeweiligen Kontrollflusskonstrukte repräsentieren dann die Grenzen eines Kontrollflussblocks (vergleiche Abbildung 9.9).

[4] $\wp\,(S)$ bezeichne die Potenzmenge von S.

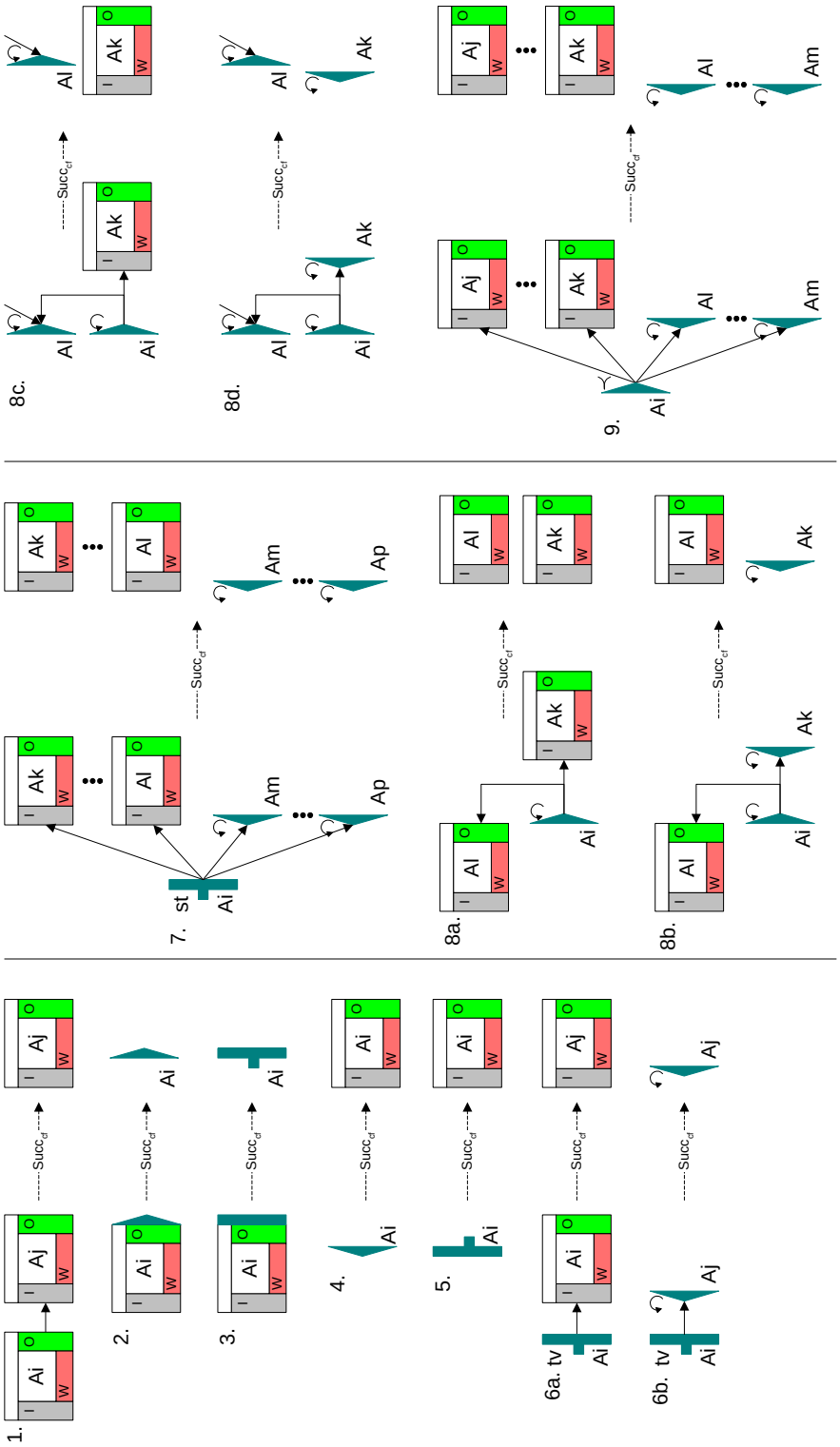



**Abbildung 9.8:** Graphische Definition der Nachfolgerrelation $Succ_{cf}$

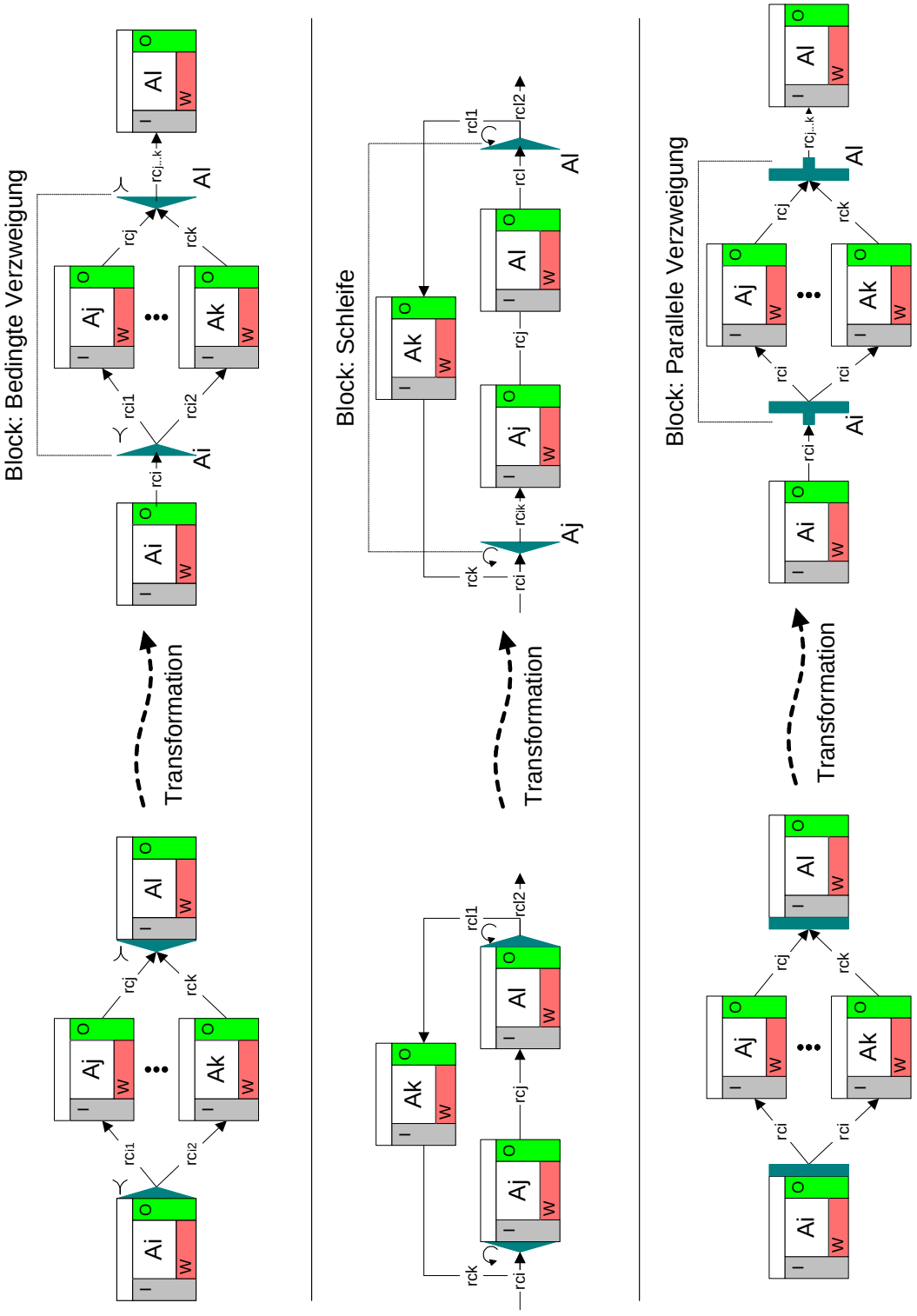



**Abbildung 9.9:** Transformation der Workflow-Graphen auf die erweiterte Aktivitätenmenge

*Definition 5: Kontrollflusskante* $Cfe$

Eine Kontrollflusskante wird durch Angabe zweier Knoten aus $ACTIVITIES^{ext}$ (Startknoten beziehungsweise Endknoten) und aus einem dem Startknoten zugeordneten Returncode spezifiziert:

$$
\begin{array}{rcl}
Cfe(WF) & : & ACTIVITIES^{ext} \times RETURNCODES \times ACTIVITIES^{ext} \\
(N_i, rc, N_j) & \in & Cfe(WF) :\Leftrightarrow \\
1. \quad N_i & \in & ACTIVITIES \ \wedge \\
N_j & \in & ACTIVITIES^{ext} \ \wedge \\
N_j & \in & N_i.Succ_{cf} \ \wedge \\
rc & \in & RETURNCODES(N_i) \\
\vee & & \\
2. \quad N_i & = & \blacktriangleright \ \wedge \\
N_j & \in & ACTIVITIES^{ext} \ \wedge \\
N_j & \in & N_i.Succ_{cf} \ \wedge \\
rc & \in & RETURNCODES(N_i.Pred_{cf}) \ \wedge \\
\exists & & cfABS = (rc, w, N_j) \in \mathbf{T}^{*}_{\mathbf{WEP}} : CFaltBranchSpec \rightarrow^{*} cfABS \qquad \circledast \\
\vee & & \\
3. \quad N_i & \in & \{ \blacktriangleleft, \dashv, \vdash \} \ \wedge \\
N_j & \in & ACTIVITIES^{ext} \ \wedge \\
N_j & \in & N_i.Succ_{cf} \ \wedge \\
rc & \in & RETURNCODES(N_i.Pred_{cf}), \qquad \text{falls } N_i.Pred_{cf} \in ACTIVITIES \\
 & & RETURNCODES(N_i.Pred_{cf}.Pred_{cf}), \text{ falls } N_i.Pred_{cf} \in \{\blacktriangleright, \dashv\}
\end{array}
$$

Die mit $\circledast$ markierte Zeile stellt sicher, dass nur vorhandene Kombinationen aus $rc$ und Nachfolgeraktivitäten gewählt werden.

Mittels der definierten Kontrollflusskanten lassen sich leicht Kontrollflusspfade ableiten. Es wird deshalb als nächstes bestimmt, welche globalen Objekte mit welchen Qualitätsstufen durch einen beliebigen vorgegebenen Kontrollflusspfad beschrieben werden.

### 9.3.2 Bestimmung der durch einen Kontrollflusspfad beschriebenen globalen Objekte

Sei $cfPath$ ein beliebiger Kontrollflusspfad eines Workflows. Gesucht wird die Menge der globalen Objekte inklusive der Qualitätsstufen der Objektversionen, die in diesen globalen Objekten durch die Aktivitäten des betrachteten Kontrollflusspfads $cfPath$ zur Laufzeit gespeichert werden. Die Qualitätsstufen repräsentieren dann entweder Zwischenergebnisse oder Endergebnisse der Aktivitäten auf diesem Kontrollflusspfad $cfPath$. Gesucht werden also Tripel der Form:

( *globales Objekt*, {*Qualitätsstufen Zwischenergebnisse* }, {*Qualitätsstufen Endergebnisse* })

Ein solches Tripel wird zukünftig als $qGlobalObject$ bezeichnet. Zur formalen Definition dieser im Folgenden als $QglobalObjects^{WF}(cfPath)$ bezeichneten Menge von Tripeln der Form $qGlobalObject$ wird noch eine weitere Definition benötigt:

Mit $QlsOfParam^{A}_{op}(rc)$ wird die Menge der Qualitätsstufen bezeichnet, welche die Objektversionen des Ausgabeparameters $op$ zur Laufzeit besitzen, wenn die Aktivität $A$ mit Returncode $rc$ beendet wird. Diese Festlegung ist notwendig, da Aktivitäten im **WEP**-Modell abhängig vom Returncode unterschiedliche Ausgabeparameter belegen können. Auch hier werden die Qualitätsstufen wieder in end- und zwischenergebnis-repräsentierende aufgeteilt. Die im höchsten Meilenstein referenzierte Qualitätsstufe stellt die Qualitätsstufe des Endergebnisses für den Returncode dar. Alle anderen Qualitätsstufen referenzieren Zwischenergebnisse.

*Definition 6:* $QlsOfParam^{A}_{op}$

$$
\begin{aligned}
&QlsOfParam^{A}_{op}: \\
&\quad RETURNCODES(A) \times \wp(OQUALITYLEVELS(Objectclass(op))) \times \\
&\quad \wp(OQUALITYLEVELS(Objectclass(op))) \\
&(oql_{med}, oql_{fin}) \in QlsOfParam^{A}_{op}(rc) :\Leftrightarrow \\
&\exists\ rc \in RETURNCODES(A), \\
&\quad ms_i, \ldots, ms_j \in AMILESTONES(A), \\
&\quad oql_i, \ldots, oql_j \in OQUALITYLEVELS(Objectclass(op)) : \\
&\quad (op, oql_i, w_i)\ IsPartialWordOf\ ms_i \wedge \\
&\qquad\qquad \vdots \\
&\quad (op, oql_j, w_j)\ IsPartialWordOf\ ms_j \wedge \\
&\quad ms_i\ IsPartialWordOf\ rc \wedge \\
&\qquad\qquad \vdots \\
&\quad ms_j\ IsPartialWordOf\ rc \wedge \\
&\quad ms_{max} := Max(ms_i, \ldots, ms_j) \wedge \\
&\quad (op, oql_{max}, w_{max})\ IsPartialWordOf\ ms_{max} \wedge \\
&\quad oql_{fin} := \{oql_{max}\} \\
&\quad oql_{med} := \bigcup oql_i \setminus \{oql_{max}\}
\end{aligned}
$$

Damit lässt sich nun die Menge $QglobalObjects^{WF}(cfPath)$ wie folgt definieren:

*Definition 7:* $QglobalObjects^{WF}(cfPath)$

$$
\begin{aligned}
&QglobalObjects^{WF}(cfPath): \\
&\quad GLOBALOBJECTS(WF) \times \wp(OQUALITYLEVELS(GLOBALOBJECTS(WF))) \\
&\quad \times \wp(OQUALITYLEVELS(GLOBALOBJECTS(WF))) \\
&(O, oql_{med}, oql_{fin}) \in QglobalObjects^{WF}(cfPath) :\Leftrightarrow \\
&\exists\ dfout_i \in DATAFLOW(WF), cfe_i \in cfPath, \\
&\quad A_i \in ACTIVITIES, A_j \in ACTIVITIES^{ext}, \\
&\quad rc_i \in RETURNCODES(A_i), O \in GLOBALOBJECTS(WF) : \\
&\quad dfout_i = (A_i, op_i, O) \wedge \\
&\quad cfe_i = (A_i, rc_i, A_j) \wedge \\
&\quad \nexists\, dfout_k \in DATAFLOW(WF), cfe_k \in cfPath, A_k \in ACTIVITIES : \\
&\quad\quad dfout_k = (A_k, \ldots, O) \wedge \\
&\quad\quad (cfe_k = (A_k, \ldots, \ldots) \vee cfe_k = (\ldots, \ldots, A_k)) \wedge \\
&\quad (oql_{med}, oql_{fin}) := QlsOfParam^{A_i}_{op_i}(rc_i)
\end{aligned}
$$

$QglobalObjects^{WF}(cfPath)$ enthält damit für jedes globale Objekt ein Tripel $qGlobalObject$, das neben dem globalen Objekt die Beschreibung der Qualitätsstufen von Objektversionen umfasst, die

von der letzten in das globale Objekt schreibenden Aktivität aus $cfPath$ als Zwischen- beziehungsweise Endergebnisse dort abgelegt werden können.

### 9.3.3 Bestimmung der durch einen Elementarblock beschriebenen globalen Objekte

$QglobalObjects^{WF}$ lässt sich mittels Schnitt- und Vereinigungsmengenbildung auf elementare Kontrollflussblöcke erweitern. Darunter versteht man Kontrollflussblöcke, in denen außer der Sequenz nicht weitere Kontrollflussblöcke enthalten sind. Im Folgenden wird $QglobalObjects^{WF}$ für jeden möglichen Elementarkontrollflussblock des **WEP**-Modells hergeleitet.

#### Elementarkontrollflussblock Parallelität

Sei $cfBlock_{par}$ ein beliebiger paralleler Kontrollflussblock und seien $cfPath_i$, ..., $cfPath_j$ die Menge der Kontrollflusspfade des Blocks. Da bei der Parallelität immer alle Zweige durchlaufen werden, wird $QglobalObjects^{WF}(cfBlock_{par})$ als die Vereinigungsmenge der $QglobalObjects^{WF}$ aller seiner Kontrollflusspfade gebildet:

$$QglobalObjects^{WF}(cfBlock_{par}) := \bigcup_{cfPath_i \in cfBlock_{par}} QglobalObjects^{WF}(cfPath_i)$$

Die entstandene Menge kann vereinfacht werden, indem Tripel, die auf das gleiche globale Objekt referenzieren (gleiche erste Komponente), sich aber bei den Qualitätsstufen (zweite und dritte Komponente) unterscheiden, mittels Vereinigungsoperation zusammengefasst werden:

Seien $qO_1 = (O, \{oql_{med_i}, \ldots, oql_{med_j}\}, \{oql_{fin_k}, \ldots, oql_{fin_l}\})$ und
$qO_2 = (O, \{oql_{med_m}, \ldots, oql_{med_n}\}, \{oql_{fin_o}, \ldots, oql_{fin_p}\})$ zwei solche Tripel. Diese Tripel können zusammengefasst werden zu:
$(O, \{oql_{med_i}, \ldots, oql_{med_j}, oql_{med_m}, \ldots, oql_{med_n}\}, \{oql_{fin_k}, \ldots, oql_{fin_l}, oql_{fin_o}, \ldots, oql_{fin_p}\})$.

Variable Parallelität ist aus Sicht der Datenflussanalyse als Spezialfall der statischen Parallelität mit nur einem parallelen Zweig pro Traversierungsmerkmal zu betrachten.

#### Elementarkontrollflussblock bedingte Verzweigung

Sei $cfBlock_{alt}$ ein beliebiger bedingter Verzweigungskontrollflussblock und seinen $cfPath_i$, ..., $cfPath_j$ die Menge seiner Kontrollflusspfade. $QglobalObjects^{WF}(cfBlock_{alt})$ wird gebildet als die Schnittmenge bezüglich der ersten Komponente der $QglobalObjects^{WF}$ aller seiner Kontrollflusspfade:

$$QglobalObjects^{WF}(cfBlock_{alt}) := \bigcap^{1}_{cfPath_i \in cfBlock_{alt}} QglobalObjects^{WF}(cfPath_i)^5$$

[5] $\bigcap^{i}$ bildet die Schnittmenge bezüglich der $i$.Komponente eines n-Tupels. Sie beinhaltet damit alle n-Tupel, die in der $i$.Komponente übereinstimmen.

$$\begin{array}{rcl} \cap^i & : & \wp(C_1 \times \ldots \times C_n) \times \wp(C_1 \times \ldots \times C_n)\ C_i \text{ beliebige Menge} \\ S_1 \cap^i S_2 & := & \{s_1 = (\ldots, s_1^i, \ldots) \in S_1 : \exists\, s_2 = (\ldots, s_2^i, \ldots) \in S_2 : s_2^i = s_1^i\} \cup \\ & & \{s_2 = (\ldots, s_2^i, \ldots) \in S_2 : \exists\, s_1 = (\ldots, s_1^i, \ldots) \in S_1 : s_1^i = s_2^i\} \end{array}$$

Die Schnittmengenbildung stellt sicher, dass nur die globalen Objekte in der Menge $QglobalObjects^{WF}(cfBlock_{alt})$ vorkommen, die auf *allen* Kontrollflusspfaden durch diesen Block beschrieben werden. Analog zum Elementarkontrollflussblock Parallelität kann die entstandene Menge mittels Vereinigungsoperation bezüglich der Komponenten zwei und drei vereinfacht werden.

#### Elementarkontrollflussblock Schleife

Bei der Bestimmung der $QglobalObjects^{WF}$-Menge für den Schleifenblock müssen im allgemeinen Fall (vergleiche Abbildung 9.10) die Kontrollflusspfade

$$\begin{aligned} cfPath_{noloop} &= {}^{\circlearrowleft}\blacktriangleleft_{A1} \rightarrow A1 \rightarrow A3 \rightarrow A4 \rightarrow {}_{A4}\blacktriangleright^{\circlearrowleft} \xrightarrow{rc2} \text{ und} \\ cfPath_{withloop} &= {}^{\circlearrowleft}\blacktriangleleft_{A1} \rightarrow A1 \rightarrow A3 \rightarrow A4 \rightarrow {}_{A4}\blacktriangleright^{\circlearrowleft} \xrightarrow{rc1} A2 \rightarrow {}^{\circlearrowleft}\blacktriangleleft_{A1} \rightarrow A1 \rightarrow A3 \rightarrow \\ & \quad A4 \rightarrow {}_{A4}\blacktriangleright^{\circlearrowleft} \xrightarrow{rc2} \end{aligned}$$

betrachtet werden. $A2$ und $A3$ repräsentieren beliebige Blockaktivitäten.

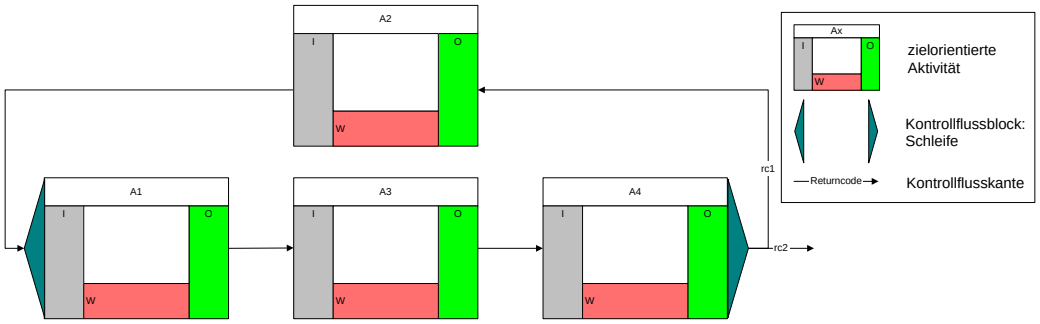



**Abbildung 9.10:** Pfade durch einen elementaren Schleifenblock

Da für jedes globale Objekt jeweils nur die letzte schreibende Aktivität eines Pfades von Belang ist, kann der Pfad $cfPath_{withloop}$ vereinfacht werden zu: $cfPath_{withloop} = A4 \rightarrow {}_{A4}\blacktriangleright^{\circlearrowleft} \xrightarrow{rc1} A2 \rightarrow {}^{\circlearrowleft}\blacktriangleleft_{A1} \rightarrow A1 \rightarrow A3 \rightarrow A4 \rightarrow {}_{A4}\blacktriangleright^{\circlearrowleft} \xrightarrow{rc2}$ Gesucht sind wiederum die globalen Objekte, die unabhängig vom Pfad mit Objektversionen versorgt werden. Es muss also eine Schnittmengenbildung bezüglich der ersten Komponente durchgeführt werden. Da bei einer Schleife der Kontrollflusspfad $cfPath_{noloop}$ immer ein Suffix-Pfad des Pfads $cfPath_{withloop}$ darstellt, liefert eine Schnittmengenoperation immer genau den Pfad $cfPath_{noloop}$. Es gilt also für die Menge $QglobalObjects^{WF}$ einer elementaren Schleife $cfBlock_{loop}$:

$$QglobalObjects^{WF}(cfBlock_{loop}) := QglobalObjects^{WF}(cfPath_{noloop}).$$

### 9.3.4 Aufbau eines kumulierten Kontrollflusspfads

Damit sind alle Voraussetzungen für die Bestimmung des kumulierten Kontrollflusspfads geschaffen, da rekursiv jeder Elementarblock durch eine Blockaktivität ersetzt werden kann, die exakt die gleichen globalen Objekte beschreibt wie dieser elementare Block: Bei dieser Ersetzung wird für jedes globale Objekt $O$ aus $QglobalObjects^{WF}(cfBlock)$ des zu ersetzenden elementaren Kontrollflussblocks $cfBlock$ ein klassenkonformer Ausgabeparameter $op_O$ definiert und mit dem globalen Objekt $O$ über

eine Datenflussausgangskante verbunden. Die Menge $QlsOfParam_{op_O}^{A_{cfBlock}}$ lässt sich aus der Menge $QglobalObjects^{WF}(cfBlock)$ für jeden beliebigen Elementarkontrollflussblock $cfBlock$– wie oben beschrieben – bestimmen (siehe Beispiel in Abbildung 9.11).

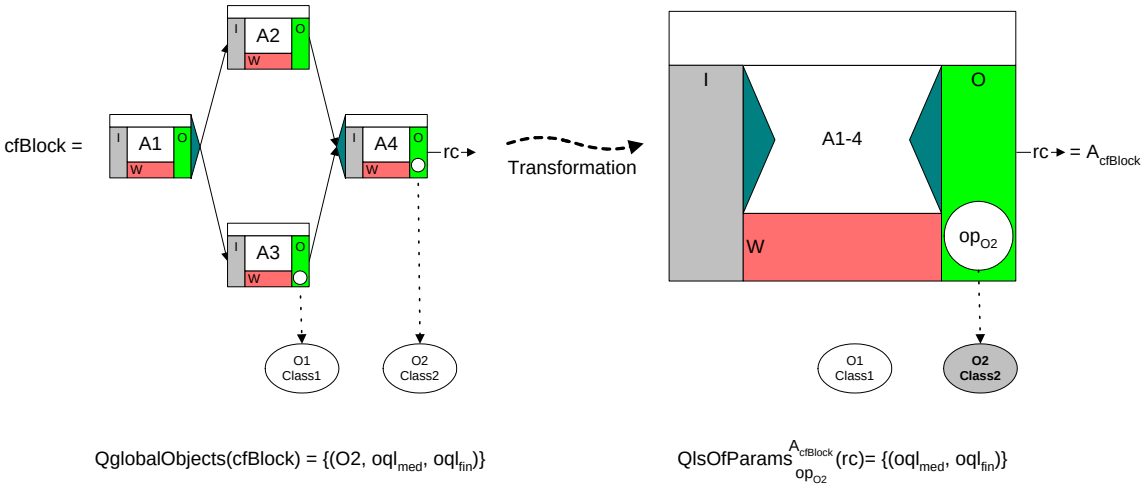



**Abbildung 9.11:** Beispiel für die Blockersetzung

Auf diese Weise können beliebig strukturierte Vorgängerblöcke auf eine Sequenz von Aktivitäten $cfPath_{pred}^{A} = A_{block_1} \rightarrow \ldots \rightarrow A_{block_n}$ zurückgeführt werden. Diese Sequenz beschreibt den kumulierten Kontrollflusspfad zur Aktivität $A$.

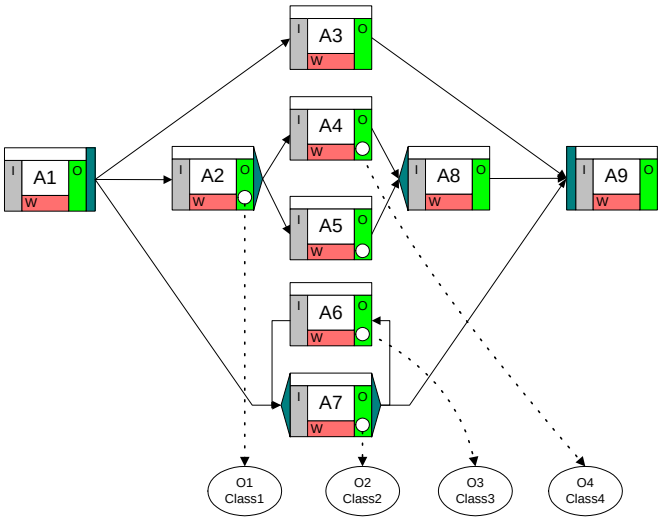
### 9.3.5 Bestimmung der durch parallele Kontrollflusspfade beschriebenen globalen Objekte

Die Analyse paralleler Kontrollflusspfade[6] erfolgt nach dem gleichen Schema. Für alle parallelen Pfade kann *ein* kumulierter Kontrollflusspfad $cfPath_{par}^{A}$ hergeleitet und für diesen die $QglobalObjects^{WF}(cfPath_{par}^{A})$-Menge bestimmt werden (vergleiche Abbildung 9.12).

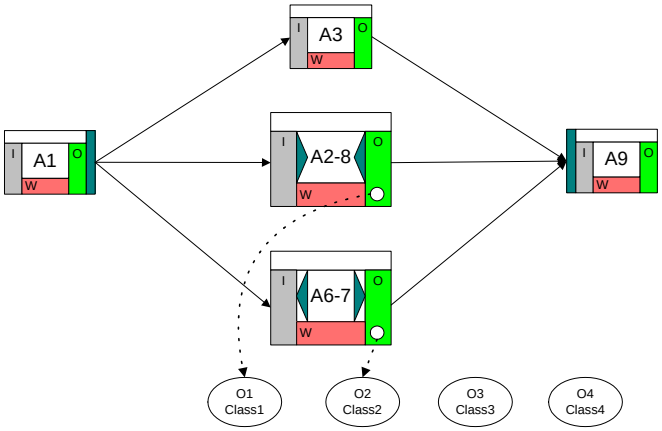
### 9.3.6 Korrekte und SE-optimierte Datenflusskanten

Für jede Aktivität $A$ eines Workflows können auf die oben beschriebene Weise die zwei Mengen $QglobalObjects^{WF}(cfPath_{pred}^{A})$ und $QglobalObjects^{WF}(cfPath_{par}^{A})$ bestimmt werden. Diese Mengen enthalten exakt die globalen Objekte, die als Startknoten für Datenflusseingangskanten bei der Aktivität $A$ für den Eingabeparameter $ip$ verwendet werden können. Neben der Klassenkonformität muss für eine *korrekte* Datenflusseingangskante gelten, dass die Endergebnisse repräsentierenden Qualitätsstufen innerhalb des Qualitätsbereichs des Eingabeparameters $ip$ liegen müssen. Für eine *SE-optimierte* Datenflusseingangskante muss der Qualitätsbereich von $ip$ auch die Qualitätsstufen der Zwischenergebnisse umfassen. Dies führt zu folgenden Definitionen:

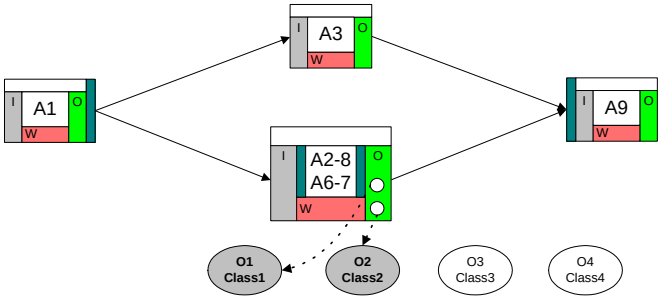
[6] Ein Algorithmus zur Bestimmung aller zu einer Aktivität parallelen Pfade wird später in Abschnitt B.1.1 vorgestellt.




(a) Ausgangssituation




(b) Ein kumulierter Kontrollflusspfad pro paralleler Zweig




(c) Ein kumulierter Kontrollflusspfad für alle parallelen Zweige

**Abbildung 9.12:** Beispiel für die Entwicklung des parallelen kumulierten Kontrollflusspfads zu $A3$

*Definition 8: korrekte und SE-optimierte Datenflusskanten*

Sei die qObjekt-Spezifikation eines globalen Objekts
$qO$ = ($O$, $oql_{med}$, $oql_{fin}$) $\in QglobalObjects^{WF}(cfPath^{A}_{pred}) \cup QglobalObjects^{WF}(cfPath^{A}_{par})$, $A \in ACTIVITIES(WF)$, $ip \in APINPUTS(A)$ und $O \in GLOBALOBJECTS(WF)$ gegeben:

- Die Datenflusseingangskante $dfin = (O, A, ip)$ ist *korrekt*, wenn gilt:

  $$\forall\, oql_{fin} \in qO.oql_{fin} : apiqRange_{min} \leq oql_{fin} \leq apiqRange_{max},$$

  wobei $apiqRange_{min}$ und $apiqRange_{max}$ die Grenzen des Datenqualitätsbereichs von $ip$ darstellen.

- Gilt außerdem:

  $$\forall\, oql_{med} \in qO.oql_{med} : apiqRange_{min} \leq oql_{med} \leq apiqRange_{max},$$

  so ist die Datenflusseingangskante $dfin$ *SE-optimiert*, da auch alle bereitgestellten Zwischenergebnisse (= Objektversionen mit Qualitätsstufen aus $oql_{med}$) von der lesenden Nachfolgeraktivität benötigt werden.

Definition 8 legt zur Modellierungszeit überprüfbare Kriterien für ein korrektes beziehungsweise SE-optimiertes **WEP**-Datenflussmodell fest. Der folgende Satz zeigt für datenflusszyklenfreie Workflow-Graphen, dass damit auch die in Definition 2 geforderten Laufzeiteigenschaften erfüllt sind.

*Satz 1: Analyse korrekter und SE-optimierter datenflusszyklenfreier **WEP**-Datenflussmodelle*

*Ein **WEP**-Datenflussmodell ist korrekt, wenn alle notwendigen Eingabeparameter aller Aktivitäten mit genau einer korrekten Datenflusseingangskante verbunden sind.*

*Es ist SE-optimiert, wenn diese Datenflusseingangskanten auch noch SE-optimiert sind.*

*Beweis:*

Satz 1 besagt, dass ein Datenflussmodell, das Definition 8 erfüllt, auch die Definition 2 einhält. Der Beweis dieser Aussage erfolgt durch Widerspruch:

Sei $dfin = (O, A, ip)$ eine beliebige Datenflusseingangskante, welche die Definition 8 erfüllt. Es gilt also:

1. Die qObjekt-Spezifikation $qO = (O, oql_{med}, oql_{fin})$ des globalen Objekts $O$ ist im kumulierten Vorgängerpfad $cfPath^{A}_{pred}$ oder im kumulierten Parallelitätspfad $cfPath^{A}_{par}$ enthalten:

   $qO \in QglobalObjects^{WF}(cfPath^{A}_{pred}) \cup QglobalObjects^{WF}(cfPath^{A}_{par})$.

2. Die Qualitätsstufen $oql_{fin}$ (und $oql_{med}$ für SE-optimierte **WEP**-Datenflussmodelle) befinden sich innerhalb des Qualitätsbereichs des betrachteten Eingabeparameters $ip$.

Es wird vorerst angenommen, dass keine qObjekt-Spezifikation $qO = (O, oql_{med}, oql_{fin})$ von $O$ in $QglobalObjects^{WF}(cfPath^{A}_{par})$ liegt: Damit muss der Parameter $ip$ von $A$ durch Vorgängeraktivitäten mit Objektversionen in den richtigen Qualitätsstufen versorgt werden. Definition 2 ist in diesem Fall nicht erfüllt, wenn ein Kontrollflusspfad von der Startaktivität des Workflows zur betrachteten Aktivität existiert, bei dem das globale Objekt $O$ nicht mit Objektversionen beschrieben beziehungsweise nur mit Objektversionen der falschen Qualitätsstufen versorgt wird. Im ersten Fall ist keine

qObjekt-Spezifikation von $O$ in $QglobalObjects^{WF}(cfPath^{A}_{pred})$ enthalten, da dort nur globale Objekte enthalten sind, die auf **allen** Vorgängerpfaden mit Objektversionen versorgt werden (vergleiche Herleitung des kumulierten Vorgängerpfads in den Abschnitten 9.3.2, 9.3.3 und 9.3.4). Zweiter Fall ist durch die Voraussetzung 2 ausgeschlossen. Damit wurde ein Widerspruch für diesen Fall herbeigeführt.

Es bleibt nun noch der Fall zu untersuchen, bei dem keine qObjekt-Spezifikation von $O$ in $QglobalObjects^{WF}(cfPath^{A}_{pred})$ enthalten ist. Um Definition 8 zu erfüllen, muss der Eingabeparameter von $ip$ aus einem parallelen Zweig versorgt werden. Es muss also für $O$ gelten: $\exists$ qObjekt-Spezifikation $qO = (O, oql_{med}, oql_{fin}) \in QglobalObjects^{WF}(cfPath^{A}_{par})$. Definition 2 ist in diesem Fall genau dann nicht erfüllt, wenn keine der Aktivitäten, die auf zur Aktivität $A$ parallelen Zweigen liegen, das globale Objekt $O$ mit geeigneten Objektversionen beliefert. Damit ergibt sich ein Widerspruch zur Voraussetzung $(O, oql_{med}, oql_{fin}) \in QglobalObjects^{WF}(cfPath^{A}_{par})$ (vergleiche Abschnitt 9.3.5) beziehungsweise zur Voraussetzung 2 des Beweises und der Satz ist für alle Fälle bewiesen.

■

### 9.3.7 Erkennung blockierender Datenflusszyklen

Satz 1 ist in obiger Form nicht für Datenflussmodelle korrekt, die Datenflusszyklen zwischen parallelen Zweigen beinhalten. Abbildung 9.13 zeigt ein Beispiel:

Qualitätsstufenüberschneidungen vorausgesetzt, sind alle in Abbildung 9.13 gezeigten Datenflusseingangskanten korrekt beziehungsweise SE-optimiert. Nach Satz 1 ist damit das Datenflussmodell auch gemäß Definition 2 korrekt beziehungsweise SE-optimiert und die Datenversorgung der Aktivitäten sichergestellt. Dies ist jedoch offensichtlich falsch. Aktivitäten $A2$, $A3$ und $A4$ blockieren sich gegenseitig, da sie auf Eingabeparameter angewiesen sind, die in den jeweils anderen Zweigen produziert werden. Es ist deshalb unabdingbar, die erlaubten korrekten beziehungsweise SE-optimierten Datenflusskanten weiter einzuschränken, um obige Datenflusszyklen zu vermeiden. Es müssen jedoch nicht alle Datenflusszyklen verhindert werden, sondern nur solche, bei denen die Datenversorgung ausschließlich über Aktivitäten aus parallelen Zweigen erfolgt. Hier kann nämlich zur Laufzeit keine Sequenzialisierung der Aktivitäten gemäß Datenabhängigkeiten gefunden werden.

Beim Einfügen einer Datenflusskante muss deshalb überprüft werden, ob ein solcher blockierender Datenflusszyklus entsteht, und gegebenenfalls das Einfügen abgelehnt werden. Der folgende Algorithmus erkennt blockierende Datenflusszyklen, in dem er sich bei jedem globalen Objekt $O$ in der Menge $ObjOfDFpath(O)$ vermerkt, welche anderen globalen Objekte bereits von $O$ über einen Datenflusspfad erreichbar sind.

#### Algorithmus „Verhinderung blockierender Datenflusszyklen“

##### Einfügen einer Datenflusseingangskante

Sei $dfin = (O, A, ip)$ eine beliebige korrekte beziehungsweise SE-optimierte Datenflusseingangskante, die von einem globalen Objekt $O$ ausgeht, in das nur Aktivitäten schreiben, die auf zu $A$ parallelen Zweigen liegen. Es gilt also für die qObjekt-Spezifikation $qO = (O, oql_{med}, oql_{fin})$ von $O$: $qO \in QglobalObjects^{WF}(cfPath^{A}_{par}) \setminus QglobalObjects^{WF}(cfPath^{A}_{pred})$. Sei $GOout(A)$ die Menge aller globalen Objekte, in die $A$ schreibt.

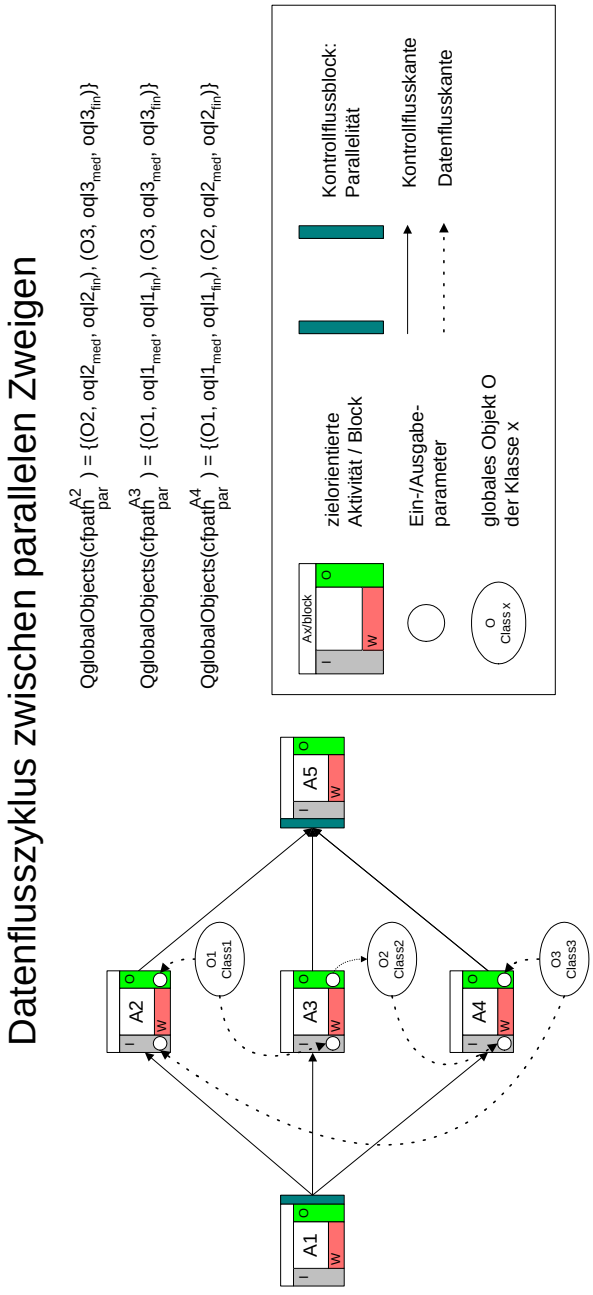



**Abbildung 9.13:** Beispiel für einen blockierenden Datenflusszyklus

- Die Datenflusseingangskante $dfin$ kann im Datenflussmodell des Workflows verbleiben, wenn keines der in $ObjOfDFpath(O)$ enthaltenen Objekte von $A$ beschrieben wird:

$$\nexists\, O_i \in ObjOfDFpath(O) : O_i \in GOout(A)$$

  Das globale Objekt $O$ wird in die $ObjOfDFpath$-Mengen aller „Ausgabeobjekte“ der Aktivität $A$ aufgenommen. Außerdem müssen auch die globalen Objekte aus $ObjOfDFpath(O)$ eingefügt werden. Somit werden auch alle indirekt gelesen Objekte erfasst.

- Andernfalls muss die Datenflusseingangskante $dfin$ entfernt werden, da sie einen Datenflusszyklus vollenden würde, der zum Blockieren des Workflows zur Laufzeit führen würde.

**Einfügen einer Datenflussausgangskante**

Sei $dfout = (A, op, O)$ eine beliebige korrekte beziehungsweise SE-optimierte Datenflussausgangskante.

Sei $GOin(A)$ eine Menge aller globaler Objekte, von denen $A$ liest.

- Die Datenflussausgangskante $dfout$ kann im Datenflussmodell des Workflows verbleiben, wenn $A$ von keinem der globalen Objekte $O_i$ aus $GOin(A)$ liest:

$$\nexists\, O_i \in ObjOfDFpath(O) : O_i \in GOin(A)$$

  Füge die $ObjOfDFpath$-Mengen aller „Eingabeobjekte“ von $A$ in die $ObjOfDFpath$-Menge des globalen Objekts $O$ ein.

- Andernfalls kann die Datenflussausgangskante nicht in das Datenflussmodell aufgenommen werden.

Die Abbildung 9.14 zeigt das Vorgehen am Beispiel der Abbildung 9.13

*Satz 2: Analyse korrekter und SE-optimierter beliebiger **WEP**-Datenflussmodelle*

*Enthält ein **WEP**-Datenflussmodell nur korrekte beziehungsweise SE-optimierte Datenflusskanten, die nach dem Algorithmus aus Abschnitt 9.3.7 eingefügt worden sind, so gilt Satz 1 auch für **WEP**-Datenflussmodelle mit Datenflusszyklen.*

*Beweis:*

Der Algorithmus lehnt das Einfügen einer Datenflusskante ab, die zu einem blockierenden Datenflusszyklus führen wurde. Es gilt damit Satz 1.

■

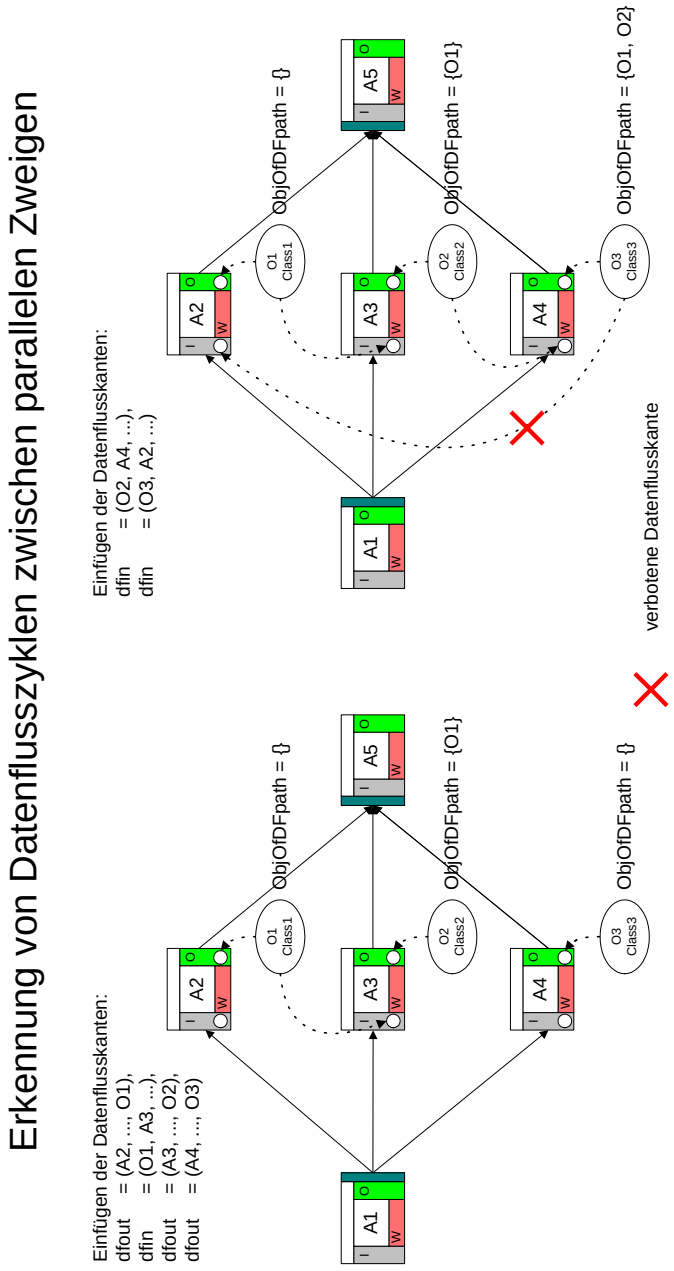



**Abbildung 9.14:** Vorgehensweise beim Finden blockierender Datenflusszyklen

# Teil III

# Die WEP-Ausführungskomponente zur Workflow-Steuerung

# Kapitel 10

# Einführung und Entwurfsziele

Workflow-Management-Systeme sind angetreten, um durch die Trennung von Ablauflogik und Anwendungsimplementierung die Modellierung *und* Steuerung realer Prozesse zu vereinfachen. Bei der Entwicklung eines Workflow-Modells wie dem **WEP**-Modell muss damit die systemgestützte Steuerung der aus dem modellierten Workflow abgeleiteten Workflow-Instanzen den gleichen Stellenwert wie die eigentliche Workflow-Modellierung besitzen. Dazu ist es notwendig, dass das entworfene Workflow-Modell um *dynamische* Aspekte erweitert werden kann. Darunter fallen insbesondere die Festlegung von Bearbeitungszuständen sowie eine operationelle Semantik, mit deren Hilfe eine Workflow-Instanz von einem konsistenten Bearbeitungszustand in einen anderen konsistenten Zustand überführt werden kann.

Durch diese Anforderungen, auch das dynamische Verhalten in einem Workflow-Modell zu berücksichtigen, unterscheiden sich Workflow-Modellierungsansätze unter anderem von Modellen aus dem Bereich der Datenmodellierung [BCN92, EN99] oder des Software-Engineerings [Par98]. Modellierungsansätze aus diesen Bereichen setzen ihre Schwerpunkte stärker in Analyse und Transformierbarkeit.

Während sich Teil II dieser Arbeit um die Beschreibung von Workflows unter der Berücksichtigung der besonderen Anforderungen von Entwicklungsprozessen beschäftigte, widmet sich Teil III der Steuerung dieser modellierten **WEP**-Workflows.

Hierbei muss auf der einen Seite festgelegt werden, welche Interaktionsmuster zwischen Prozessbeteiligten und dem **WEP**-Workflow-Management-System angeboten werden müssen, um die im Teil I skizzierten Anforderungen aus dem Engineering-Anwendungsgebiet (Unterstützung unstrukturierter Teilprozesse, prozesskoordiniertes Simultaneous-Engineering und dynamische Prozessstrukturanpassung) gerecht zu werden (Kapitel 11). Bei einem System, das vorzeitige Datenweitergabe erlaubt, muss auch ein adäquates und eindeutiges Systemverhalten beim Eintreffen aktuellerer Daten definiert werden, das für alle Prozessbeteiligten nachvollziehbar ist. Diese im **WEP**-Modell genannten *Konsistenzsicherungsphasen* sind Gegenstand von Kapitel 12.

Auf der anderen Seite ist die interne Verarbeitungslogik zu formalisieren. Hier liegt das Hauptaugenmerk auf der Behandlung der Eigenschaften, durch die sich der **WEP**-Ansatz von anderen Workflow-Modellen unterscheidet. Die Verarbeitungslogik von **WEP**-Workflows wird in Kapitel 13 in Form eines formalen Ausführungsmodells detailliert spezifiziert. Das **WEP**-Ausführungsmodell bildet die

Grundlage für den Nachweis dynamischer Eigenschaften von **WEP**-Workflow-Instanzen in Kapitel 14.

Durch die Konzepte der vorzeitigen Datenweitergabe und der Traversierung komplex strukturierter Objekte unterscheidet sich das **WEP**-Workflow-Management-System signifikant von anderen Workflow-Management-Systemen. Die Umsetzung der in Teil II eingeführten Konzepte zur Laufzeit wird ausführlich in den Kapiteln 15 und 16 betrachtet.

Vorzeitige Datenweitergabe ermöglicht Simultaneous-Engineering-Phasen, in denen neben der asynchronen Koordination der Prozessbeteiligten wie in klassischen Workflow-Management-Systemen auch synchrone Interaktionen zwischen den Beteiligten möglich sind (Kapitel 17).

Begonnen wird dieser Teil mit der Zusammenfassung der wichtigsten Entwurfsziele für eine **WEP**-Ausführungskomponente.

Der Entwurf der **WEP**-Ausführungskomponente orientiert sich im Wesentlichen an den folgenden Zielsetzungen:

*Sicherstellung der Datenkonsistenz :*
: Trotz unstrukturierter Teilprozesse, vorzeitiger Datenweitergabe, dynamischer Parallelität und komplex strukturierter Objektversionen muss jederzeit eine konsistente Sicht auf alle Daten durch die **WEP**-Ausführungskomponente gewährleistet werden.

*Korrektheit der Ausführung:*
: Korrekt modellierte **WEP**-Workflows müssen nach endlicher Ausführungszeit immer in einem Endzustand terminieren unabhängig von den möglichen Benutzerinteraktionen[1] und Ausprägungen der in einem Workflow gelesenen, veränderten und erzeugten Daten. Ein Bearbeitungszustand eines Workflows muss sich dabei über die Bearbeitungszustände seiner Aktivitäten, die Zustände der Objektversionen in den globalen Objekten und die Belegung seiner Kanten definieren lassen. Natürlich müssen diese Zustände auch die **WEP**-Konzepte der vorzeitigen Datenweitergabe und der Konsistenzsicherungsphasen widerspiegeln. Basierend auf den Bearbeitungszuständen eines Workflows muss für jedes **WEP**-Kontrollflusskonstrukt und jede Benutzerinteraktion eine klare und eindeutige Ausführungssemantik definiert werden.

  Bearbeitungszustände und Ausführungssemantik erlauben die Definition eines *formalen **WEP**-Ausführungsmodells*, das alle erlaubten Zustandsübergänge beinhaltet. Dieses Ausführungsmodell ermöglicht die Analyse des dynamischen Verhaltens einer Workflow-Instanz und somit auch eine formale Aussage über ihre dynamischen Eigenschaften.

[1] Voraussetzung dafür sind natürlich faire Bearbeiter. Ein Workflow wird immer blockieren, wenn die Bearbeiter die ihnen zugewiesenen Aktivitäten nicht ausführen.

# Kapitel 11

# WEP-Benutzerinteraktionen zur Unterstützung von Simultaneous-Engineering

Die Forderung nach vorzeitiger Datenweitergabe zur Unterstützung von prozesskoordiniertem Simultaneous-Engineering (siehe Teil I) impliziert neue Interaktionsformen für Bearbeiter zielorientierter Aktivitäten. Die **WEP**-Benutzerinteraktionen müssen dem Aktivitätenbearbeiter ermöglichen, vor Beendigung ihrer Aktivität Datenobjekte an nachfolgende Aktivitäten weiterzugeben, vorzeitig weitergereichte Objektversionen zurückzunehmen, neuere Objektversionen bei Vorgängeraktivitäten zu erhalten beziehungsweise selbst anzufordern und Konsolidierungsrunden einzuberufen, in denen an einem Objekt beteiligte Personen über eine neue Version entscheiden können. Weiterhin schließt die Verwendung eines objektorientierten Datenmodells Interaktionen zum Erzeugen von Objekt- und Subobjektversionen ein.

## 11.1 Die WEP-Interaktionsmetapher

Ein **WEP**-Laufzeitklient muss diese zusätzlichen Interaktionsmöglichkeiten entsprechend bereitstellen. Als sinnvollste Metapher für einen **WEP**-Laufzeitklienten stellte sich die Schreibtischmetapher heraus [Kno99]. Entsprechend der Schreibtisch-Metapher befinden sich alle Eingabeparameter im Eingangskorb einer Aktivität, der im **WEP**-Workflow-Management-System *Eingabebereich (input area)* genannt wird (vergleiche Abbildung 11.1). In analoger Weise liegen im Ausgangskorb – beim **WEP**-Workflow-Management-System mit *Ausgabebereich (output area)* bezeichnet – alle Ausgabeparameter. Im Ausgangskorb ist auch die Beschreibung der Meilensteine (vergleiche Abschnitt 6.2.2.2) zu finden, die – vereinfacht ausgedrückt – die noch abzuliefernden Ergebnisse repräsentieren. In der Schreibtischmitte befindet sich die eigentliche Arbeitsfläche des Schreibtisches. Sie besteht aus dem *Arbeitsbereich (work area)* und aus Referenzen auf die Schrittprogramme der Aktivität (vergleiche Abschnitt 6.2.3). Nur Objektversionen, die sich im Arbeitsbereich befinden, können über die Schrittprogramme manipuliert werden, die beim Aufruf mit den entsprechenden Objektversionen versorgt werden.

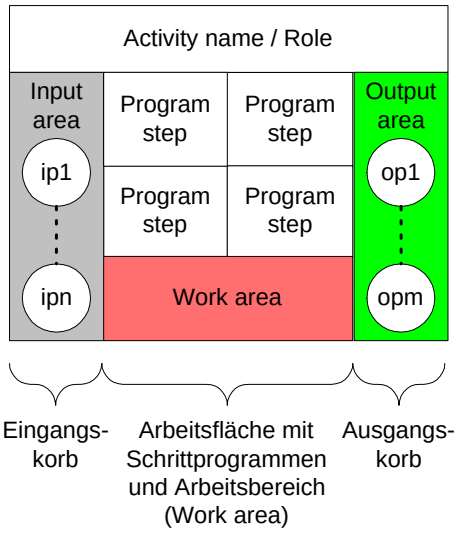



**Abbildung 11.1:** Schreibtisch-Metapher einer zielorientierten Aktivität im **WEP**-Workflow-Management-System

## 11.2 Exemplarisches Interaktionsszenario

Im Folgenden soll eine typische Interaktionsreihenfolge skizziert werden, um damit den Leser anschaulich an die Bearbeitungszustände von **WEP**-Workflows und ihre komplexe Schaltlogik heranzuführen. Danach definiert Kapitel 13 Benutzerinteraktionen, Bearbeitungszustände und Schaltlogik mit allen Bedingungen formal. Ausgangspunkt für die informelle Einführung ist das in Abbildung 11.2 dargestellte Workflow-Modell, das eine bedingte Verzweigung und zwei globale Objekte zeigt.

Abbildung 11.3 zeigt den Bearbeitungszustand des Workflows aus Abbildung 11.2 unmittelbar nach dessen Instanzierung. Die erste Aktivität des Workflows befindet sich im Zustand $offered$. Sie wird deshalb möglichen Prozessbeteiligten zur Bearbeitung angeboten. Alle anderen Aktivitäten befinden sich im Initialzustand $notOffered$.

Globale Objekte sind mit einer „Wurzel“-Objektversion belegt, die durch ihren Zustand $unset$ anzeigt, dass das globale Objekt noch keine gültige Objektversion beherbergt. Ausnahmen bilden nur die globalen Objekte, die mit den Eingabeparametern des Workflows verknüpft sind. Sie enthalten für alle notwendigen Eingabeparameter Objektversionen mit gültigen Qualitätsstufen im Bearbeitungszustand $released$. Im Beispiel gilt dies für das globale Objekt $O1$.

Datenflusskanten, die zur Modellierungszeit logischerweise nur *globale Objekte* mit Aktivitätenparametern verbinden können (vergleiche Abschnitt 8.2), verknüpfen zur Laufzeit die jeweils sichtbaren *Objektversionen* mit den Aktivitätenparametern. Ein globales Objekt fungiert zur Laufzeit somit als Container für alle seine Objektversionen. Die Sichtbarkeit einer Objektversion hängt von ihrem Zustand sowie vom Bearbeitungszustand der erzeugenden beziehungsweise lesenden Aktivitäten ab. Die Sichtbarkeit von Objektversionen wird im Detail in Kapitel 15 besprochen.

Mit der Operation $StartActivity$ kann eine Aktivität im Zustand $offered$ von einem Bearbeiter gestartet werden. Dabei wechselt sie in den Bearbeitungszustand $activated$ (vergleiche Abbildung 11.4). Anderen Bearbeitern, denen sie auch angeboten wurde, wird sie aus ihren Arbeitslisten entzogen.

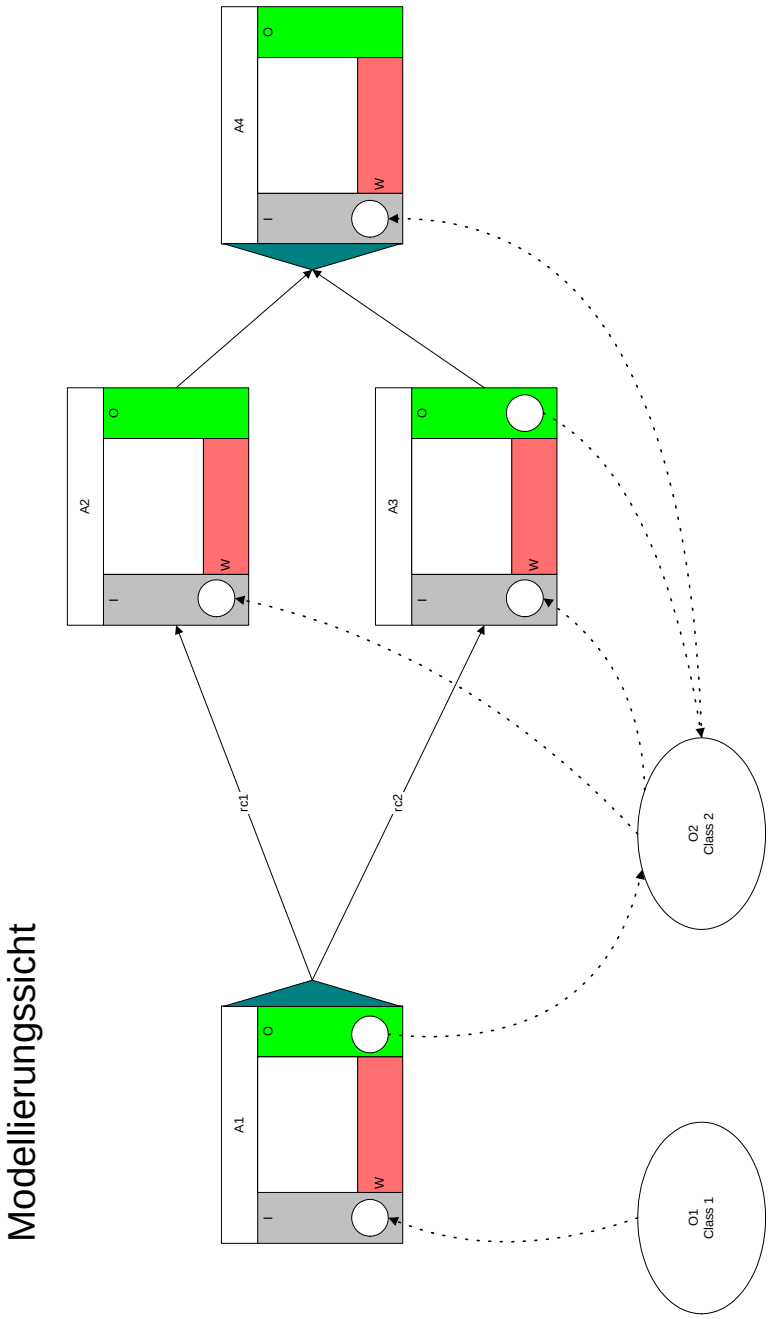



**Abbildung 11.2:** Interaktionsformen beim **WEP**-Workflow-Management-System am Beispiel

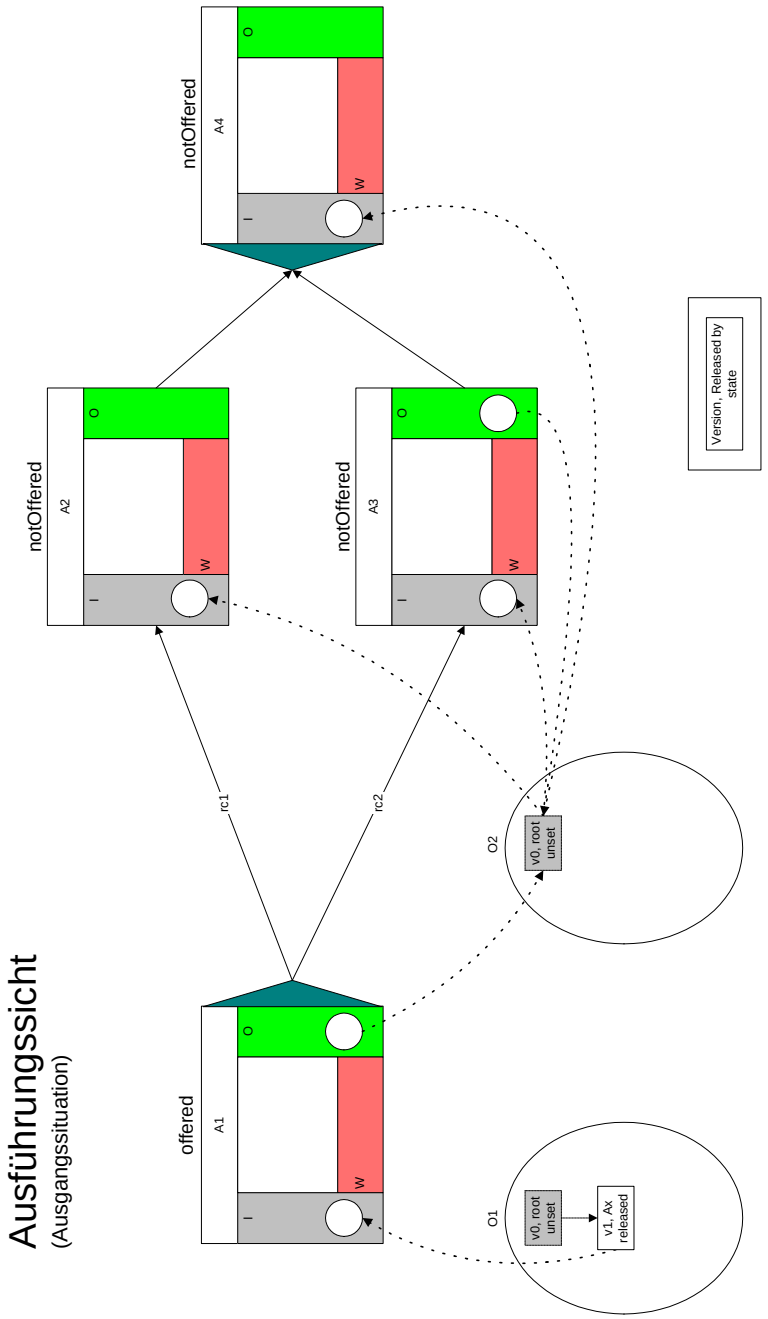



**Abbildung 11.3:** Interaktionsformen beim Start eines **WEP**-Workflows

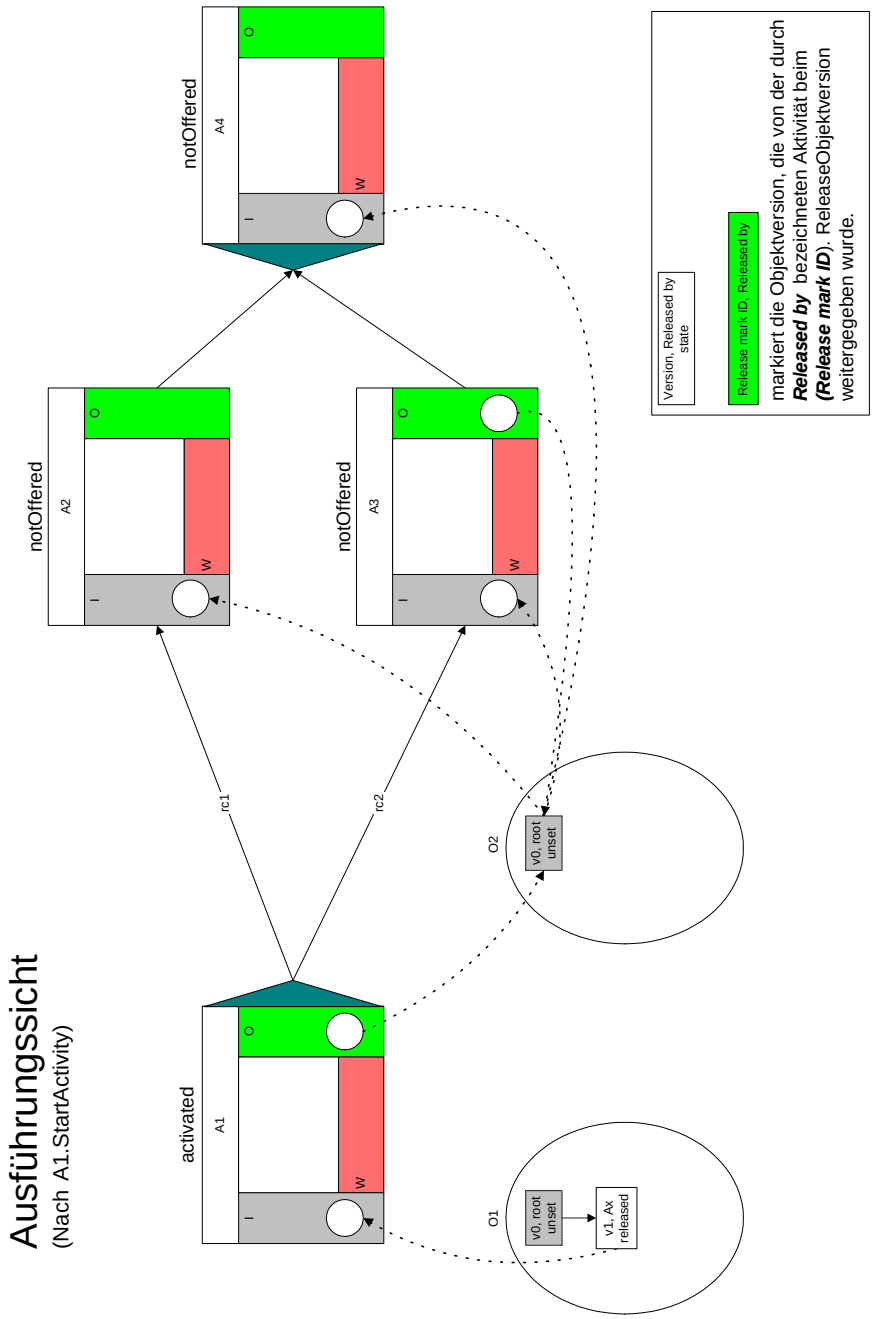



**Abbildung 11.4:** Interaktionsform $StartActivity$

Der Start einer Aktivität hat im Gegensatz zu anderen Workflow-Management-Systemen noch keine Auswirkungen auf die Objektversionen im Eingabebereich, die deshalb auch in ihrem ursprünglichen Bearbeitungszustand verbleiben und für andere Aktivitäten zugreifbar sind. Grundsätzlich gilt, dass im **WEP**-Workflow-Management-System auf alle Objektversionen im Ein- und Ausgabebereich immer nur lesend zugegriffen werden kann. Eine Veränderung ist nur durch Ableiten einer neuen Objektversion mittels der Benutzeroperation $FetchObjectVersion$ möglich. $FetchObjectVersion$ kann auf Objektversionen aus dem Eingabebereich (IN-, INOUT-Parameter) beziehungsweise Versionen im Ausgabebereich (OUT-Parameter) angewandt werden. Eine mittels $FetchObjectVersion$ erzeugte Objektversion wird im Arbeitsbereich der Aktivität angezeigt. Alle Objektversionen im Arbeitsbereich befinden sich im Bearbeitungszustand $locked$. Sie sind für Bearbeiter anderer Aktivitäten nicht sichtbar. Abbildung 11.5 zeigt einen solch skizzierten Bearbeitungszustand für die Objekte $O1$ und $O2$.

Befindet sich dagegen die Objektversion bereits im privaten Arbeitsbereich eines Bearbeiters, so muss statt $FetchObjectVersion$ die Operation $CreateObjectVersion$ zum Ableiten neuer Objektversionen angewandt werden.

Zusätzlich existieren, wie bei versionsbasierten Datenmodellen üblich [KRS98], noch Operationen zur Manipulation des Objektversionsstrukturbaums, wie beispielsweise das Erzeugen und Löschen von Subobjektversionen und das Festlegen von Konfigurationen. Diese Aspekte wurden detailliert in [Kno99] erarbeitet. Zusammen mit diesen Operationen realisieren $FetchObjectVersion$ und $CreateObjectVersion$ die im ersten Teil der Arbeit geforderten Interaktionsmöglichkeiten auf einem versionsbasierten Objektmodell.

Bei anderen Workflow-Management-Systemen werden erst mit Beendigung eines Bearbeitungsschritts die Daten an Folgeschritte weitergereicht. Besteht jedoch wie im Produktentwicklungsbereich die Notwendigkeit, während der Bearbeitung eines langdauernden Prozessschritts Daten weiterzugegeben, um zur Verkürzung der Prozesslaufzeiten Simultaneous-Engineering zu ermöglichen, so müssen für den Bearbeiter zwei getrennte Operationen für Datenweitergabe und Schrittbeendigung angeboten werden. Aus diesem Grund werden im **WEP**-Workflow-Management-System zwei eigenständige Operationen, $ReleaseObjectVersion$ für die Datenweitergabe und $FinishActivity$ für das Beenden einer Aktivität angeboten.

In Abbildung 11.6 wird die Benutzerinteraktion $ReleaseObjectVersion$ gezeigt: Für die Durchführung dieser Operation ist es notwendig, dass eine Objektversion eine in einem Meilenstein spezifizierte Qualitätsstufe erreicht hat. Der Aktivitätenbearbeiter kann dann über die Operation $ReleaseObjectVersion$ diese Objektversion in den Ausgabebereich transferieren. Dabei wird der Bearbeitungszustand der Objektversion von $locked$ auf $preReleased$ wechseln. Weitergegebene Objektversionen werden mit einem ***ReleaseMarker*** markiert, in dem die Anzahl aller $ReleaseObjectVersion$-Operationen auf einem globalen Objekt vermerkt sind. Objektversionen in diesem Zustand sind für alle nachfolgenden oder parallelen Aktivitäten sichtbar, die Datenflusseingangskanten zu dem globalen Objekt besitzen (siehe Kapitel 15). In Abbildung 11.6 ist dies für die Aktivitäten $A2$ und $A3$ der Fall. Es werden deshalb Datenflusskanten von der weitergegebenen Objektversion $v1$ zu den jeweiligen Eingabeparametern der lesenden Aktivitäten, für die diese Objektversion sichtbar ist, erzeugt.

Bei der Datenweitergabe wird noch kein Returncode gesetzt. Der Aktivitätenbearbeiter kann im Allgemeinen in diesen Phasen noch nicht abschätzen, mit welchen Ergebnis – spezifiziert durch Auswahl eines Returncodes – er die Aktivität später beenden will. Sind wie im Beispiel mehrere verschiede-

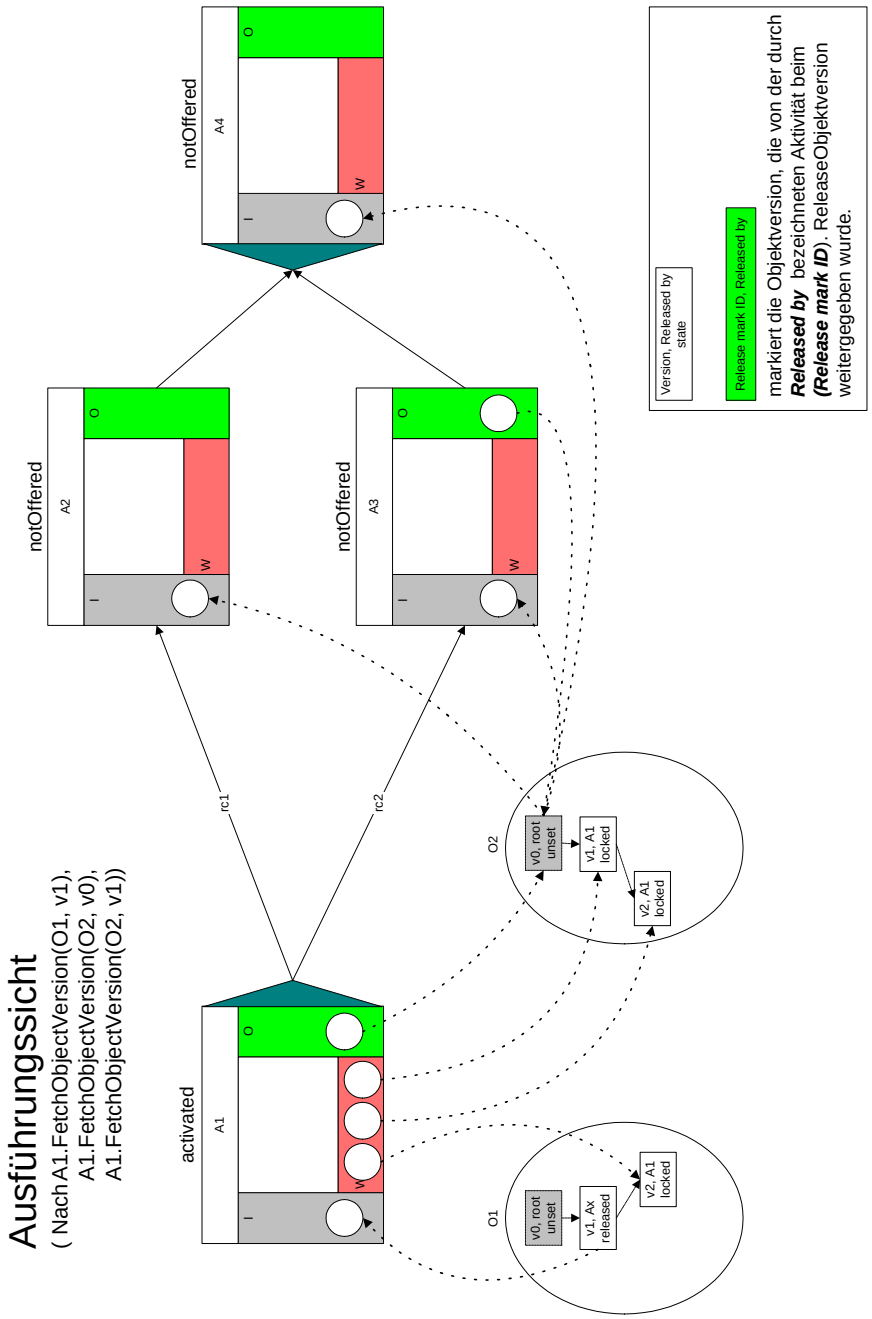



**Abbildung 11.5:** Interaktionsform $FetchObjectVersion$

ne Returncodes vorhanden, so werden die Daten deshalb über *alle* Kontrollflusspfade weitergegeben, unabhängig davon, mit welchem Returncode die Aktivität später beendet wird. Diese Vorgehensweise hat den Vorteil, dass dadurch der höchste Parallelitätsgrad und damit die kürzesten Prozessdurchlaufzeiten erzielt werden, da der durch den Returncode später gewählte „richtige“ Pfad sicher dabei ist. Es werden allerdings auch alle „falschen“ Pfade bearbeitet, die mit Beendigung der Aktivität zurückgesetzt werden müssen. [1]

Sind durch die Datenweitergabe für eine Aktivität alle notwendigen Eingabeparameter versorgt und wurden alle Vorgängeraktivitäten gestartet oder können aufgrund eines anderen bestrittenen Pfades nicht mehr aktiviert werden, so wird sie in den Bearbeitungsstand $preOffered$ versetzt (siehe $A2$ und $A3$ in Abbildung 11.6).

Da eine solche Aktivität von den Eingabedaten noch nicht beendeter Aktivitäten abhängt und sich die Daten damit noch ändern können, nennt man sie eine *abhängige* Aktivität. Alle Bearbeitungszustände abhängiger Aktivitäten werden mit dem Präfix $pre$ gekennzeichnet. Dies führt zu folgenden Definitionen von abhängigen und unabhängigen Aktivitäten:

*Definition 9: abhängige Aktivität*

Eine Aktivität ist *abhängig*, wenn gilt:

- Mindestens einer ihrer notwendigen Eingabeparameter liegt noch nicht in endgültiger Form vor. Sein Wert kann sich also noch verändern.

  oder

- Eine ihrer Vorgängeraktivitäten ist noch nicht endgültig beendet.

*Definition 10: unabhängige Aktivität*

Bei einer unabhängigen Aktivität muss gelten:

- Die Werte ihrer Eingabeparameter werden nicht mehr verändert.

  und

- Es gibt keine Vorgängeraktivitäten, die noch bearbeitet werden.

Herkömmliche Workflow-Management-Systeme kennen nur unabhängige Aktivitäten. Wie in Abbildung 11.7 zu sehen, können abhängige Aktivitäten auf Basis der vorläufigen Eingabeobjekte analog zu unabhängigen Aktivitäten gestartet werden. Auch abhängige Aktivitäten können Objektversionen ableiten, manipulieren sowie – wie in Abbildung 11.8 demonstriert – an Folgeaktivitäten weiterleiten und beendet werden. Dadurch können Aktivitäten bereits bearbeitet werden, bevor ihre Vorgänger endgültig beendet sind.[2] Abhängige Aktivitäten realisieren somit das in Teil I geforderte Simultaneous-Engineering entlang der Prozesskette und tragen damit bei, die Laufzeiten von Entwicklungsprozessen signifikant zu verkürzen.

[1] Dieser Weiterleitungsalgorithmus muss für die Praxis sicherlich noch durch geeignete Heuristiken (zum Beispiel: Wahrscheinlichkeit für Pfad $\geq 20\%$), (ausschließliche) Versorgung kritischer Pfade, Verfügbarkeit der Mitarbeiter oder durch Analysen der einem Returncode zugeordnete Meilensteine (siehe Abschnitt 6.2.2.3) verfeinert werden. Für die im Folgenden angestellten Betrachtungen ist dieser Ansatz jedoch völlig ausreichend.

[2] Darin wird deutlich, dass eine Kontrollflusskante zwischen zwei Aktivitäten eine *Start-Start*-Abhängigkeit spezifiziert. Bei herkömmlichen Workflow-Modellen wird mit einer Kontrollflusskante dagegen immer eine *Ende-Start*-Beziehung zwischen Aktivitäten modelliert.

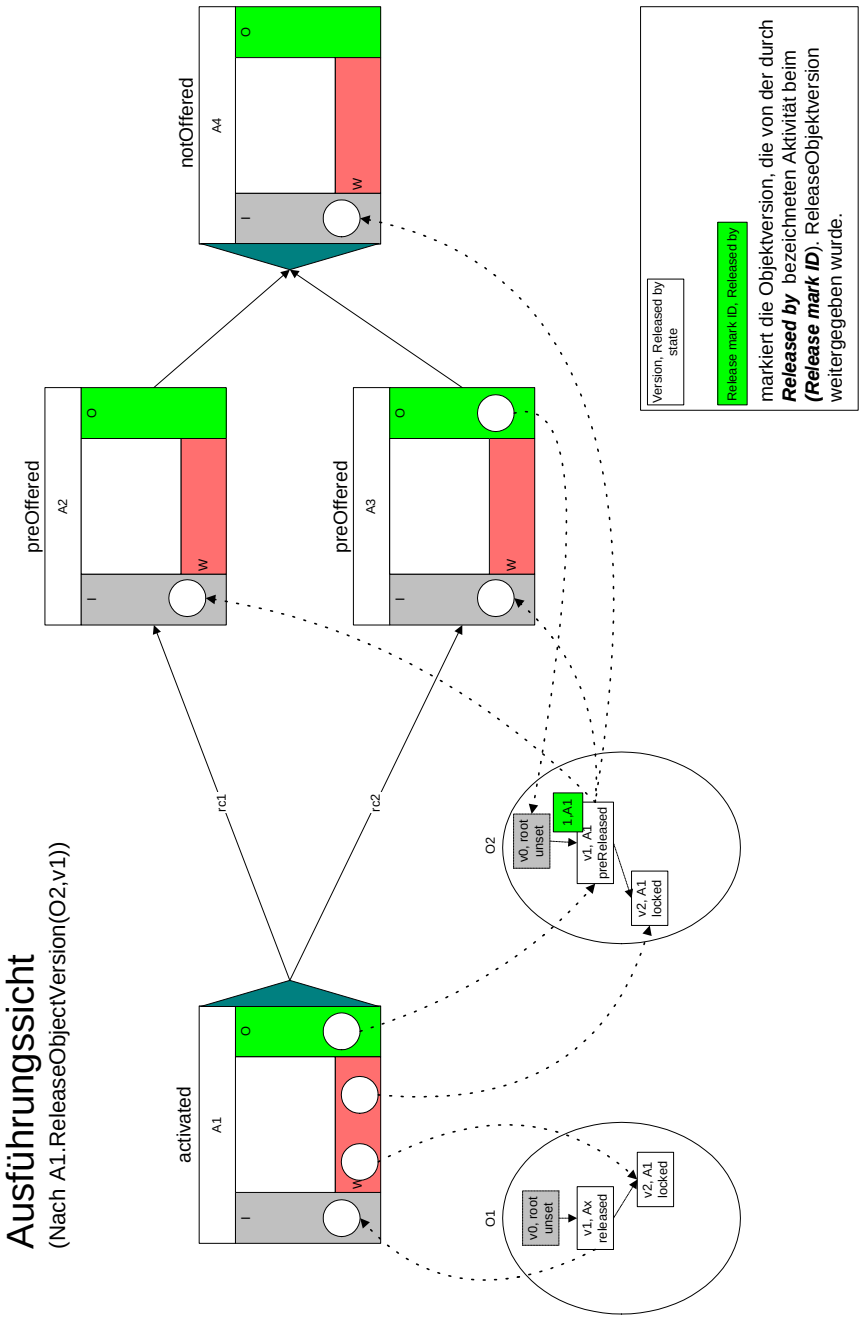



**Abbildung 11.6:** Interaktionsform $ReleaseObjectVersion$

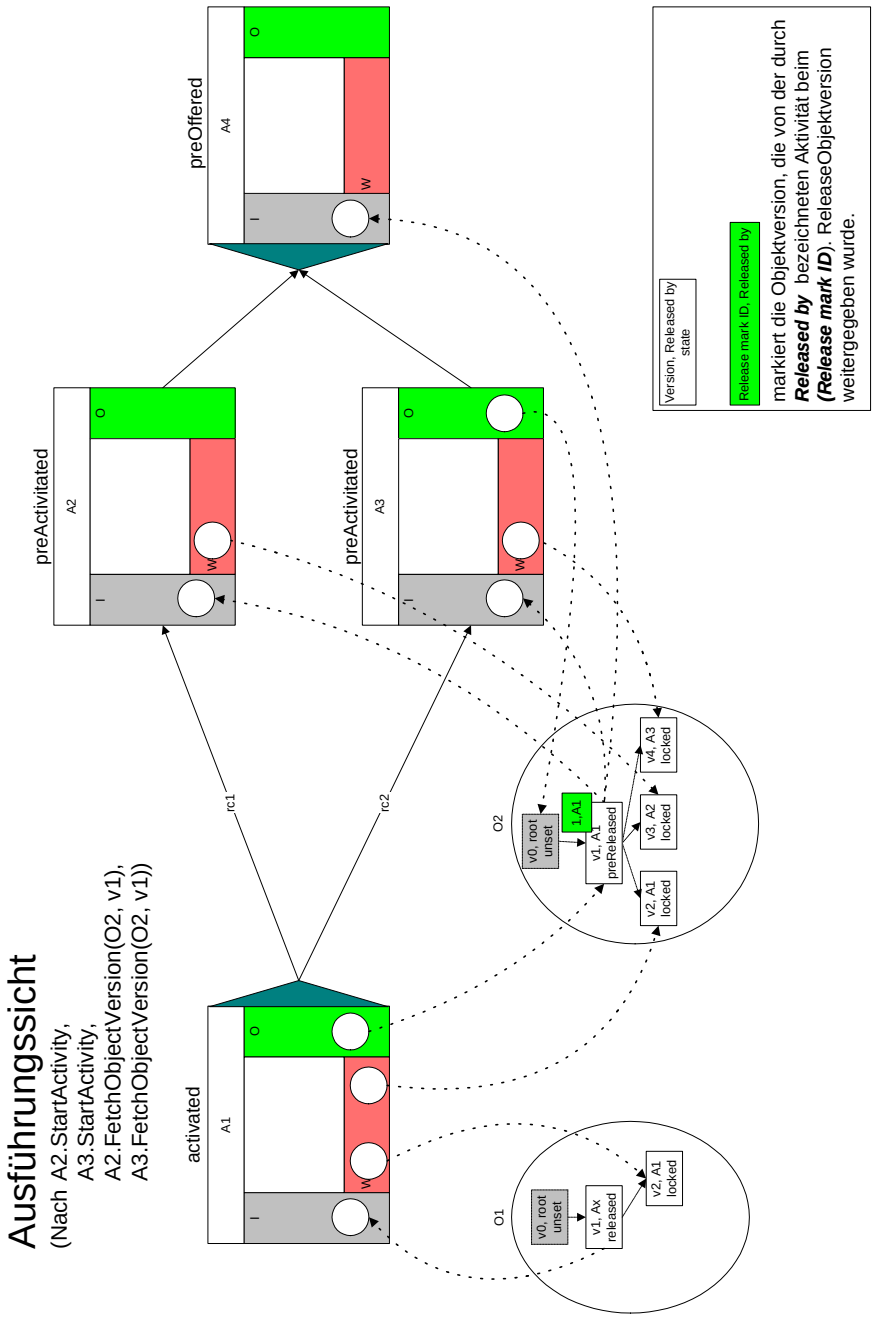



**Abbildung 11.7:** Interaktionsformen beim Starten abhängiger Aktivitäten

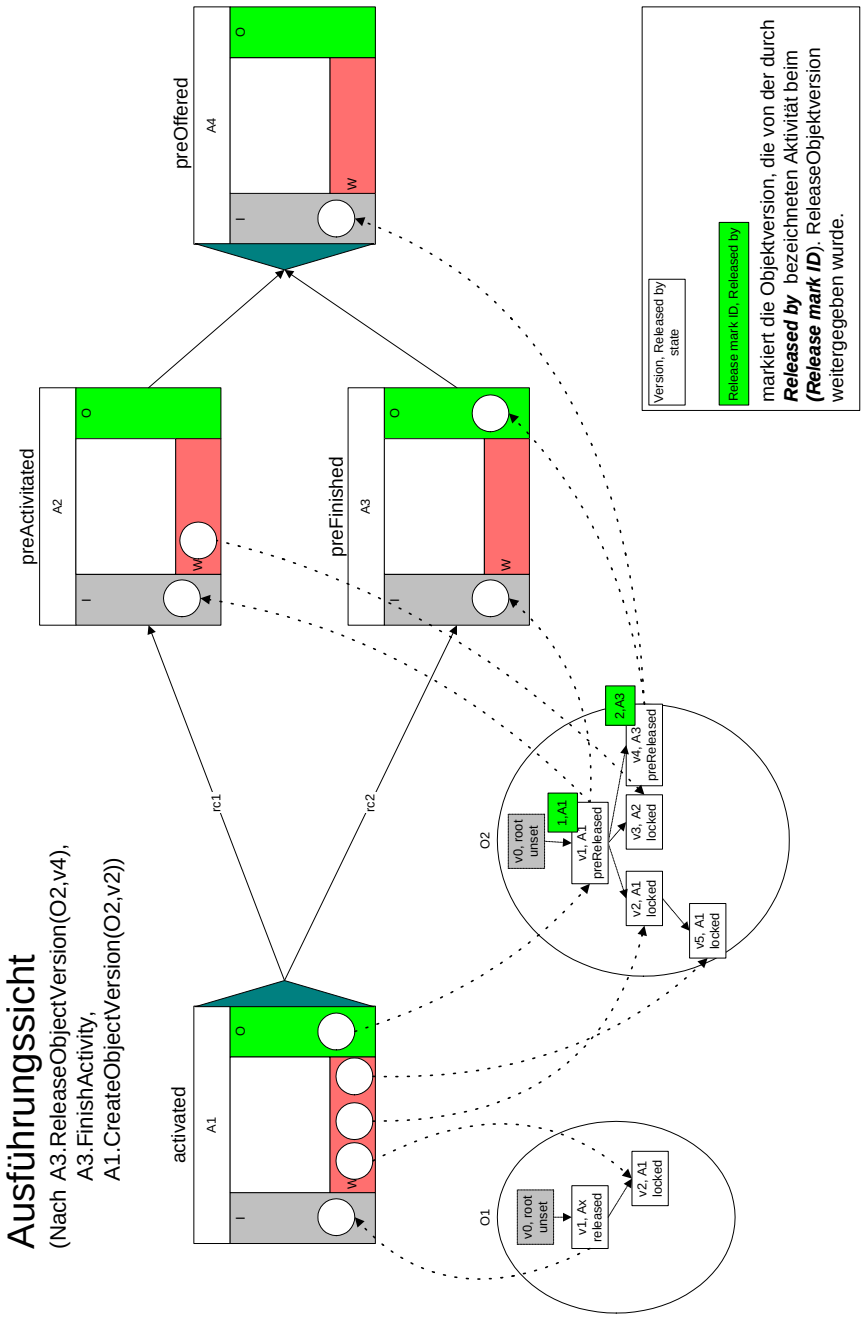



**Abbildung 11.8:** Interaktionsformen beim Beenden abhängiger Aktivitäten

Erhalten abhängige Aktivitäten neue Eingabedaten – wie in Abbildung 11.9 mit der Objektversion $v5$ des globalen Objekts $O2$ für die Aktivitäten $A2$ und $A3$ dargestellt, so wird für jede betroffene abhängige Aktivität eine Konsistenzsicherungsphase angestoßen (siehe Kapitel 12), in der unter anderem durch Aktualisierung der bisherigen Eingabeparameter wieder ein konsistenter Bearbeitungszustand aller Aktivitäten und damit auch des gesamten Workflows hergestellt wird. Während einer Konsistenzsicherungsphase befindet sich eine Aktivität im Bearbeitungszustand $preConsRecov$ [3].

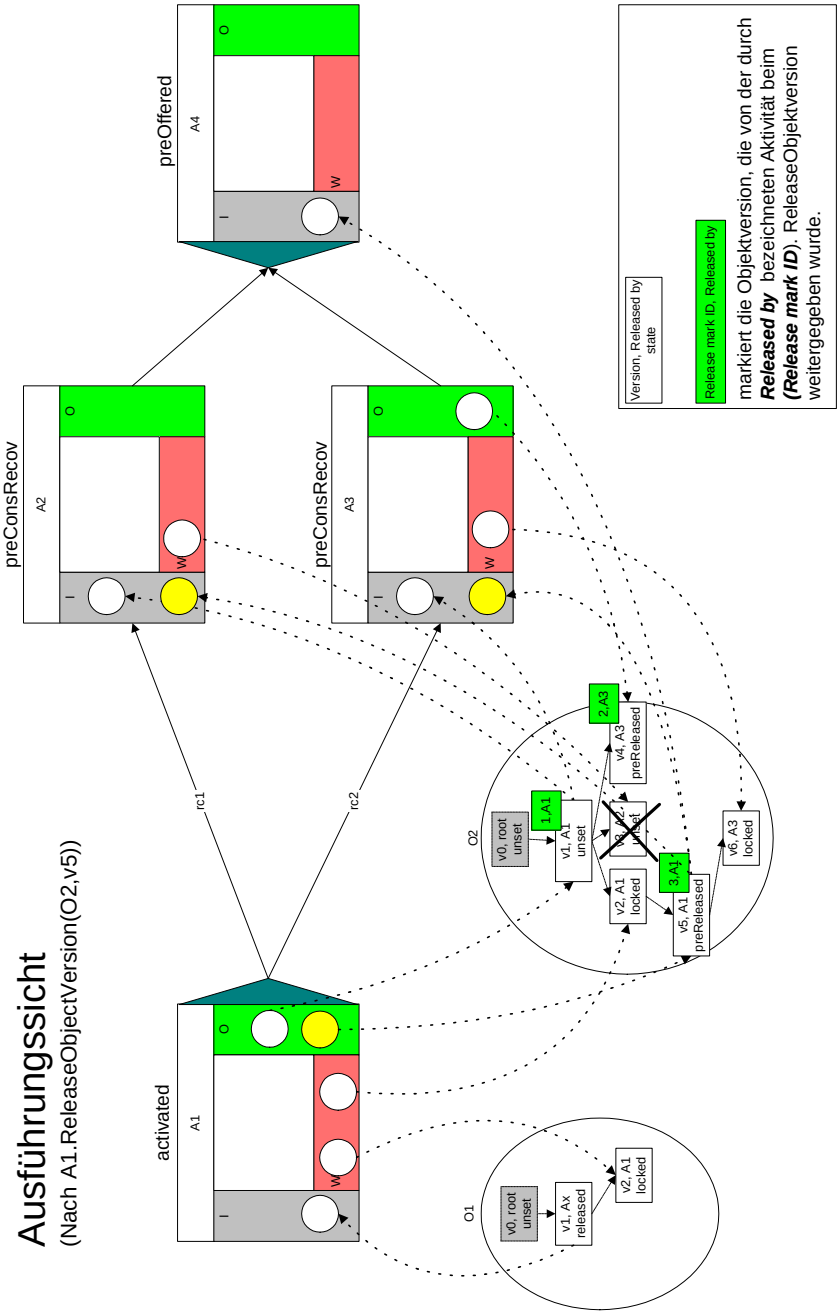



**Abbildung 11.9:** Interaktionsformen während einer Konsistenzsicherungsphase

[3] $preConsRecov$ steht für *Consistency Recovering*.

Abbildung 11.10 zeigt die Bearbeitungszustände der abhängigen Aktivitäten $A2$ und $A3$ nach Durchführung der Konsistenzsicherungsphase, signalisiert durch die Ereignisse $evUndoCompleted$ und $evMergeCompleted$. Im gezeigten Beispiel wird bei der Aktivität $A2$ das UndoRedo-Konsistenzsicherungsverfahren (siehe Kapitel 12) angewandt, bei dem alle auf Basis der alten Eingabedaten erzeugten Objektversionen als ungültig markiert und gelöscht werden. Bei Aktivität $A3$ wird dagegen das Merge-Verfahren verwendet, bei dem neue und alte Objektversion zu einer neuen konsistenten Version zusammenzugefügt werden. Aktivität $A2$ befindet sich danach folgerichtig im Bearbeitungszustand $preOffered$ und $A3$ im Bearbeitungszustand $preActivated$.

Wird eine unabhängige Aktivität durch die Operation $FinishActivity$ beendet, so werden alle Objektversionen im Arbeitsbereich gelöscht. Die Objektversionen im Ausgabebereich wechseln in den Zustand $released$, da sie nun endgültig sind. Davon abhängige Aktivitäten werden dann unabhängig, wenn alle ihre Eingabeparameter nicht mehr verändert werden können und ihre Vorgängeraktivitäten nicht mehr aktivierbar sind (vergleiche Definition 10). Diese Situation ist in Abbildung 11.11 zu sehen. Die Zustände von $A2$ und $A3$ verlieren hier ihr Präfix $pre$. Beide Aktivitäten werden also unabhängig und die skizzierte Benutzerinteraktionsfolge kann sich mit $A3$ als Startaktivität wiederholen.

Bei herkömmlichen Workflow-Management-Systemen würde dagegen erst durch $FinishActivity$ die Bearbeitung der Aktivitäten $A2$ oder $A3$ beginnen können. Je nach Länge der Simultaneous-Engineering-Phase kann im **WEP**-Modell durch vorzeitige Datenweitergabe bereits ein beträchtlicher Bearbeitungsvorsprung erreicht werden.

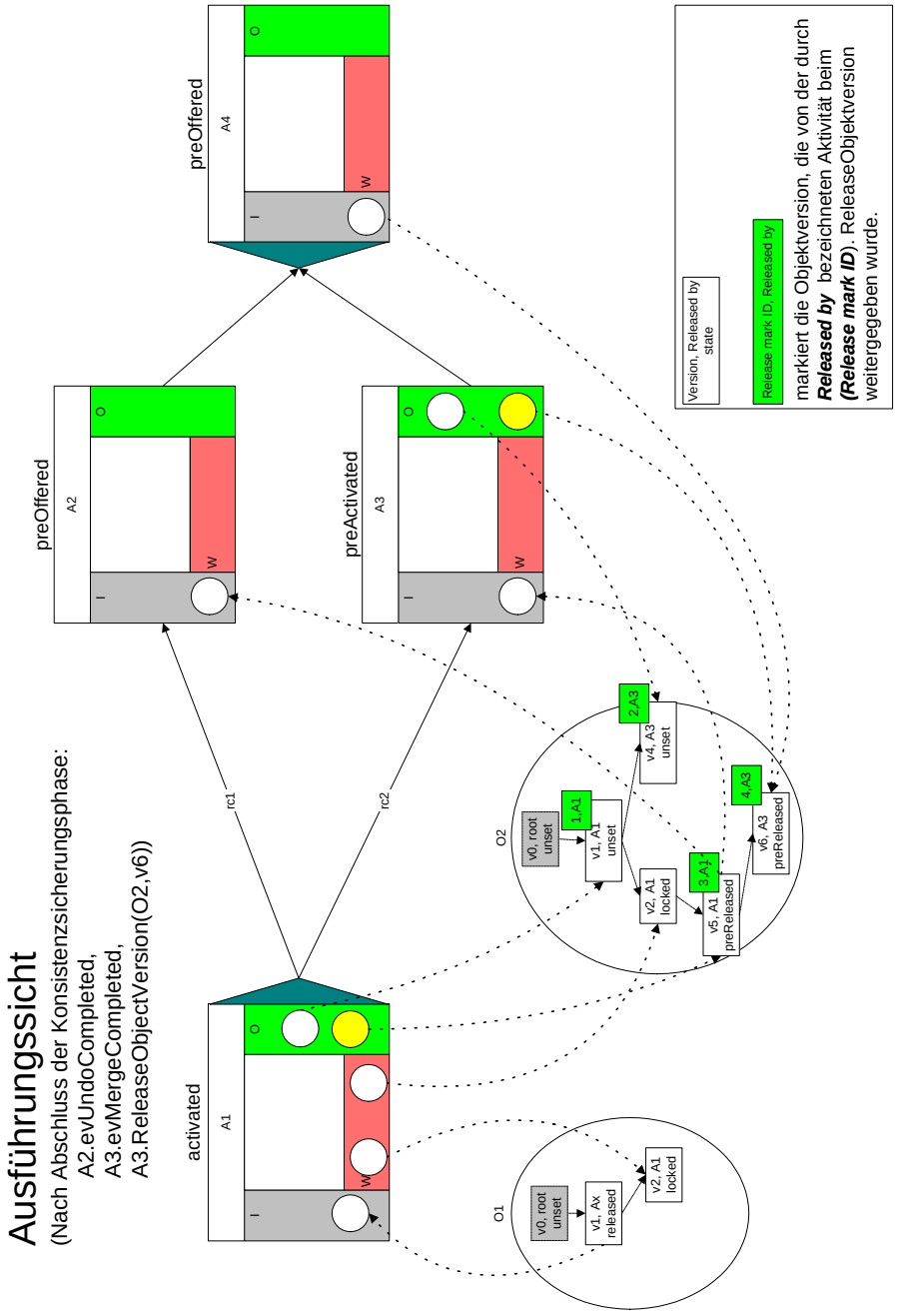



**Abbildung 11.10:** Interaktionsformen nach Abschluss einer Konsistenzsicherungsphase

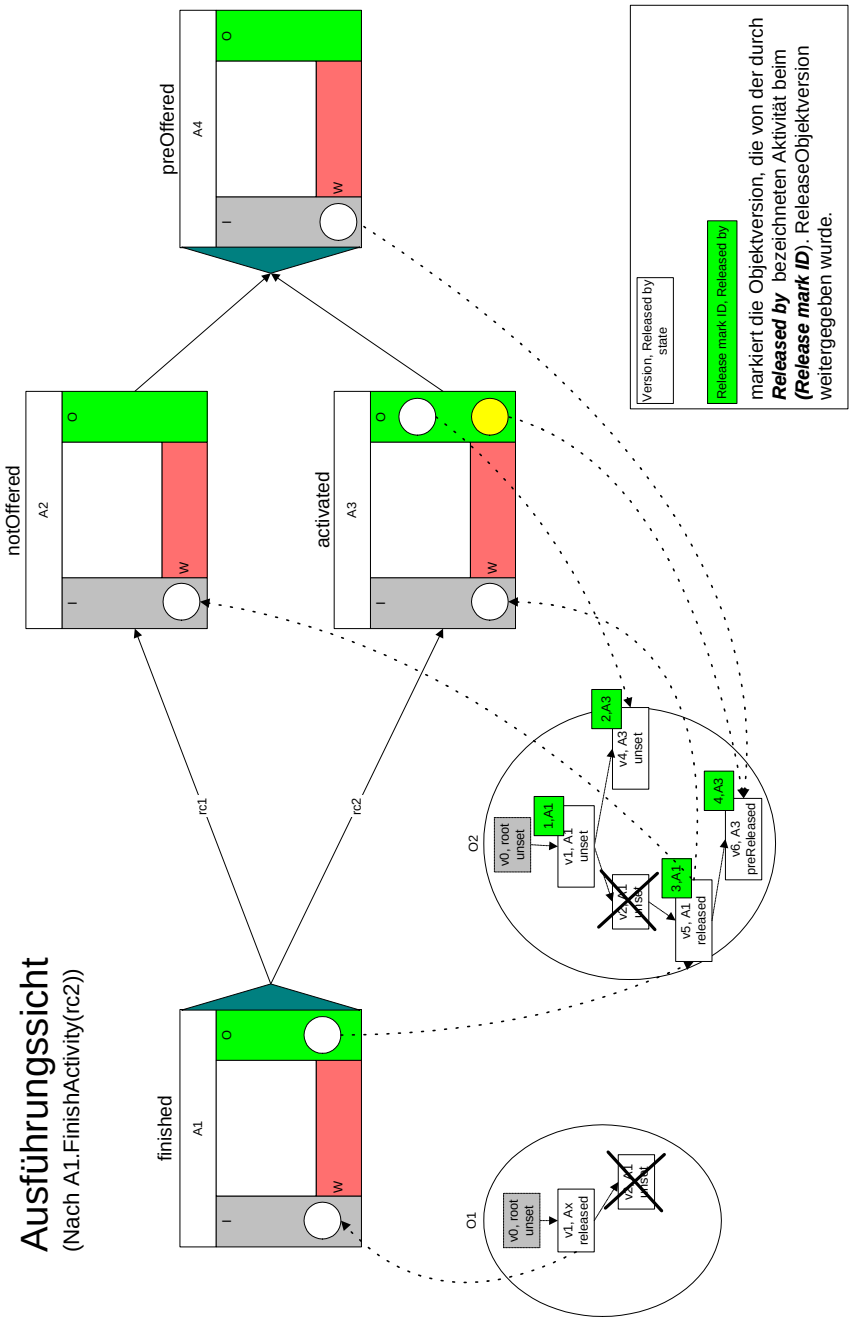



**Abbildung 11.11:** Interaktionsformen beim Wechsel in unabhängigen Bearbeitungszustand

# Kapitel 12

# Konsistenzsicherung während Simultaneous-Engineering-Phasen

Simultaneous-Engineering-Phasen reduzieren die Prozessdurchlaufzeiten nur dann, wenn die Datenkonsistenz trotz vorzeitiger Datenweitergabe gewährleistet wird. Eine Gefährdung der Datenkonsistenz kann auftreten, wenn

- aktuellere Eingabedaten bei einer Aktivität eintreffen und diese dort nicht geeignet berücksichtigt werden,
- ein anderer Kontrollflusspfad eingeschlagen wird, ohne irrtümlich beschrittene Kontrollflusspfade zurückzusetzen.

Das **WEP**-Workflow-Management-System muss deshalb geeignete Mechanismen zur Datenkonsistenzsicherung bereitstellen, die in den folgenden Abschnitten detaillierter vorgestellt werden.

## 12.1 Konsistenzsicherung beim Eintreffen aktuellerer Eingabedaten

Wie bereits im vorherigen Kapitel exemplarisch gezeigt, müssen aktuellere Eingabedaten bei einer Aktivität möglichst automatisch in die bisher geleistete Arbeit integriert werden. Dazu werden im **WEP**-Modell zwei prinzipielle Verfahren bereitgestellt, mit deren Hilfe die Datenkonsistenz mit vertretbaren Aufwand wiederhergestellt werden kann.

### 12.1.1 Das UndoRedo-Konsistenzsicherungsverfahren

Ein naheliegender Ansatz die Datenkonsistenz zu sichern, besteht darin, die Aktivität zu beenden, semantisch zurückzusetzen und mit den (nun hoffentlich) endgültigen Eingabedaten erneut zur Bearbeitung anzubieten. Damit wird sichergestellt, dass die zuletzt erzeugten Ausgabedaten einer Aktivität auf endgültigen Eingabedaten beruhen. Dies entspricht vereinfacht dem operationellen Korrektheitsbegriff für korrekt ausgeführte **WEP**-Workflow-Instanzen, der in Abschnitt 14.1 unter dem Namen SE-Serialisierbarkeit noch formal eingeführt wird.

Zur Realisierung dieses **WEP**-Konsistenzsicherungsansatzes können Kompensationsverfahren aus dem Bereich erweiterter Transaktionskonzepte [Elm92, GMS87, GMGK+91, RW91, WR90, PK88, Leu91, ANRS92] geeignet adaptiert werden, was teilweise in transaktionsorientierten Workflow-Modellen bereits getan wird [VE92, EL95, Ley95, Ley97].

Bei transaktionsorientierten Workflow-Management-Systemen wird zur Modellierungszeit zu jedem Workflow-Schritt ein sogenannter *Kompensationsschritt* definiert, der dann ausgeführt wird, wenn der bereits ausgeführte oder noch in Bearbeitung befindliche Workflow-Schritt nicht mehr benötigt wird. Es liegt in der Verantwortung des Workflow-Modellierers oder des Schrittprogrammierers einen geeigneten Kompensationsschritt zu bestimmen, der alle internen und externen Auswirkungen des Workflow-Schritts semantisch rückgängig macht (*kompensiert*). Beispiele hierfür sind die Rücküberweisung einer nicht mehr benötigten Banktransaktion oder die Stornierung einer Hotelbuchung. Die Kompensation kann dabei abhängig von der Aufgabe und des Bearbeitungszustands des Workflow-Schritts sehr komplex oder – bei nicht reversiblen Operationen – teilweise gar nicht möglich sein [Elm92].

Überträgt man die Idee der Kompensation auf das **WEP**-Modell, so könnte das **WEP**-Workflow-Management-System beim Eintreffen neuer Eingabedaten bei einer Aktivität eine zuvor modellierte Kompensationsaktivität aufrufen, welche die bisherigen Arbeiten der Aktivität automatisch semantisch rückgängig macht.

Wegen der Komplexität und der langen Bearbeitungszeiten zielorientierter Aktivitäten kann jedoch nur in den wenigsten Ausnahmefällen eine geeignete Kompensationsaktivität bereitgestellt werden, die eine vollständige Kompensation der bisherigen Auswirkungen automatisch durchführt. Glücklicherweise ist aus Anwendungssicht eine solche vollständige Kompensation der Aktivitäten in der Regel gar nicht notwendig. Aufgrund der starken Datenzentrierung genügt es, die erzeugten abhängigen Objektversionen als ungültig zu markieren (*Undo*). Dies hat weiterhin den Vorteil, dass gar keine explizite Kompensationsaktivität modelliert werden muss. Das **WEP**-Workflow-Management-System kennt von sich aus die von einer Aktivität erzeugten Objektversionen und kann sie damit automatisch als ungültig markieren. Die betroffene Aktivität wird anschließend mit den aktuellen Eingabedaten erneut zur Bearbeitung angeboten (*Redo*).

Ein erheblicher Nachteil dieser *UndoRedo-Konsistenzsicherungsmethode* bleibt trotzdem bestehen: Aufgrund der langen Bearbeitungszeiten (bis zu mehreren Monaten) kann durch die Kompensation ein Großteil bereits geleistete Arbeit „vernichtet“ werden.

Die UndoRedo-Kompensationsmethode ist für interaktive Aktivitäten deshalb nur bedingt geeignet und sollte nur in Ausnahmefällen, bei kurzen oder bei automatischen Prozessschritten eingesetzt werden.

Damit kann die UndoRedo-Konsistenzsicherungsmethode nicht das breite Spektrum zielorientierter Aktivitäten abdecken. Aus diesem Grund wurde beim **WEP**-Modell noch ein zweites Konsistenzsicherungsverfahren, das *Merge-Verfahren*, vorgesehen (vergleiche Merkmal $APIconsistencyPolicy$ in Abschnitt 6.2.1).

### 12.1.2 Das Merge-Konsistenzsicherungsverfahren

Beim Merge-Konsistenzsicherungsverfahren wird auf Kompensation verzichtet. Es wird versucht, von der durchgeführten Arbeit möglichst viel zu „retten“, ohne die Datenkonsistenz zu gefährden. Durch

den beim **WEP**-Modell verwendeten versionsbasierten Ansatz auf einem objektorientierten Datenmodell kann das **WEP**-Workflow-Management-System einem Aktivitätenbearbeiter schon sehr detaillierte Angaben darüber machen, wo sich neue und alte Eingabeobjektversionen unterscheiden. Der Bearbeiter kann dann individuell entscheiden, ob und welche Auswirkungen die neuen Eingabedaten für seine bisher geleistete Arbeit haben und entsprechend seine Ausgabeobjekte anpassen.

Diese Vorgehensweise entspricht schon heute vielfach der praktizierten Arbeitsweise in den Entwicklungsbereichen. Anwender vermissen bisher jedoch IT-Unterstützung beim Informieren und beim Vergleich von Eingabeobjektversionen (siehe Anforderungen in Kapitel 2). Beide Aspekte werden im Rahmen von Merge-Konsistenzsicherungsverfahren automatisiert. Details zur technischen Umsetzung sind in Abschnitt 13.6.2 zu finden.

Das Merge-Konsistenzsicherungsverfahren stellt auch sicher, dass immer nur Objektversionen weitergegeben werden dürfen, die von der aktuellsten Eingabeobjektversion (höchster ReleaseMarker) abgeleitet wurden. Damit sind divergierende Varianten ausgeschlossen [Kat90]. Eine Ausnahmebehandlung ist jedoch bei vorzeitiger Datenweitergabe innerhalb von Schleifen und zwischen parallelen Zweigen erforderlich. Da hier ein „Feedback“ anderer Bearbeiter auf bereits weitergegebene Ausgabeobjektversionen in der Praxis ausdrücklich gewünscht wird, dürfen in Ausnahmefällen auch von älteren Objektversionen neue Versionen abgeleitet werden. Diese Ausnahmefälle werden im Rahmen der Diskussion der Anforderungen an eine konsistente vorzeitige Datenweitergabe (siehe Kapitel 15.1) ausführlich erläutert.

Erste Erfahrungen in Anwendungsprojekten [KFW01, SAF+01] zeigen, dass beide Konsistenzsicherungsverfahren zusammen eine adäquate Lösung für die Integration neuer und alter Eingabedaten bei praxisrelevanten Produktentwicklungsprozessen darstellen.

## 12.2 Konsistenzsicherung bei anderer Kontrollflusspfadauswahl

Wie schon mit dem Ablaufszenario aus Kapitel 11 exemplarisch gezeigt wurde, werden für minimale Prozessdurchlaufzeiten bei bedingten Verzweigungen (und auch bei Schleifen) *alle* alternativen Zweige aktiviert.[1] Erst wenn der Returncode feststeht, ist eine Bearbeitung der nicht mit dem ausgewählten Returncode beschrifteten Kontrollflusszweige mit Sicherheit nicht mehr sinnvoll und die bereits erzeugten und nun nicht mehr benötigten Objektversionen müssen als ungültig markiert werden (semantisches Rollback). Ansonsten besteht die Gefahr, dass Aktivitäten mit nicht mehr gültigen Daten bearbeitet werden, wie aus dem Beispiel in der Abbildung 12.1 deutlich wird. Bei dem dort gezeigten Bearbeitungszustand des Workflows würde die Aktivität $A4$ auf Basis eines Eingabedatums ($v2$) bearbeitet werden, das von Aktivität $A2$ aus dem nicht ausgewählten Kontrollflusszweig erzeugt wurde. Die Objektversion $v2$ muss deshalb als ungültig markiert werden. Aktivität $A4$ muss stattdessen mit der Objektversion $v1$, die von der Aktivität $A3$ aus dem durch $A1$ gewählten richtigen Kontrollflusspfad erzeugt wurde, versorgt werden.

Zu erwähnen bleibt, dass aufgrund von Datenabhängigkeiten zwischen parallelen Zweigen auch Objektversionen betroffen sein können, die von Aktivitäten außerhalb der zurückgesetzten bedingten Zweige erzeugt wurden. (vergleiche Beispiel aus Abbildung 13.13 in Abschnitt 13.5.2). Diese durch das **WEP**-Workflow-Management-System automatisch angestoßene Konsistenzsicherung wird *Undo-Konsistenzsicherungsmethode* genannt und ist bis auf den Wegfall der *Redo-Phase* mit dem UndoRedo-Konsistenzsicherungsverfahren identisch.

[1] Das genaue Schaltverhalten aller Kontrollflusskonstrukte wird in Abschnitt 13.5 formal spezifiziert.

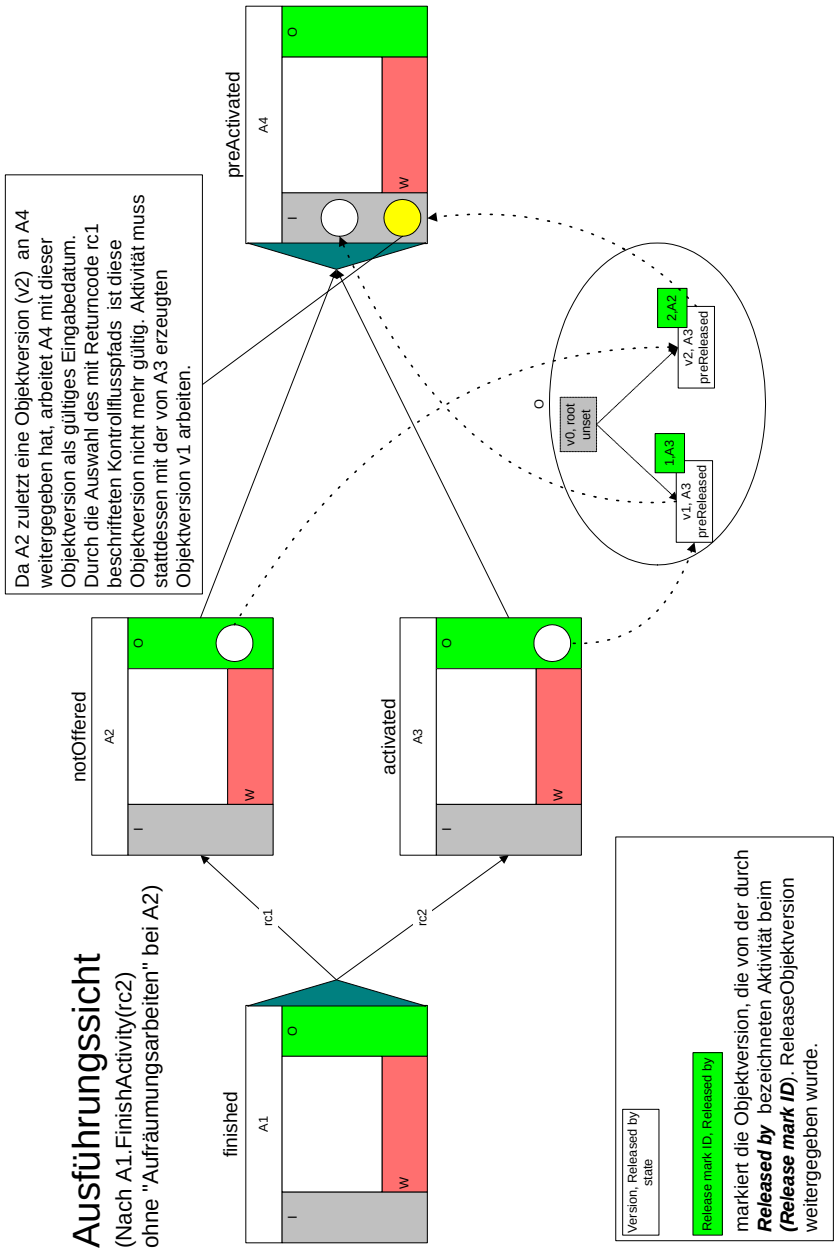



**Abbildung 12.1:** Unkorrekte Eingabedaten aufgrund der Auswahl eines anderen Kontrollflusspfads

# Kapitel 13

# Formales WEP-Ausführungsmodell

Die im Kapitel 11 exemplarisch skizzierten Benutzerinteraktionen und die Durchführung von Konsistenzsicherungsphasen (Kapitel 12) führen zu Veränderungen des Bearbeitungszustands einer **WEP**-Workflow-Instanz. Dieses Schaltverhalten eines **WEP**-Workflows soll nun verallgemeinert und formalisiert werden. Ziel ist es, das Systemverhalten in Form eines formalen **WEP**-Ausführungsmodells präzise zu beschreiben, um einerseits dem Leser einen Einblick in die Abarbeitung beliebiger **WEP**-Workflow-Instanzen zu geben und anderseits Aussagen über dynamische Eigenschaften von **WEP**-Workflow-Instanzen, wie die eine korrekte Terminierung, formulieren und nachweisen zu können.

Ein formales Ausführungsmodell kann auch als Implementierungsspezifikation herangezogen werden. Dieser Aspekt ist nicht Schwerpunkt des Kapitels. Er wird jedoch nicht außer acht gelassen. Alle **WEP**-Benutzerinteraktionen und die daraus resultierende Ereignisverarbeitung bei Kontrollflussknoten und Objektversionen werden in Form von *UML-Aktivitätsdiagrammen (UML Activity Diagrams)* [OMG01] beschrieben, die – geeignete Entwicklungsumgebungen vorausgesetzt – eine automatisierte Generierung von Code-Skeletten ermöglichen [BRJ98]. Zum Teil wird der interessierte Leser jedoch auf den Anhang verwiesen und innerhalb des Kapitels wird nur eine zusammenfassende Darstellung verwendet.

Bei der Beschreibung des formalen **WEP**-Ausführungsmodells konzentriert sich dieses Kapitel auf die Aspekte, die das **WEP**-Ausführungsmodell von den Ausführungsmodellen anderer Workflow-Management-Systeme unterscheidet und die für den Nachweis dynamischer Eigenschaften (Kapitel 14) von **WEP**-Workflow-Instanzen benötigt werden. Besondere Beachtung finden deshalb die folgende Aspekte:

- Die Möglichkeit zur vorzeitigen Datenweitergabe bedingt, dass zwischen unabhängigen und abhängigen Aktivitäten unterschieden werden muss (vergleiche Kapitel 11). Vorzeitige Datenweitergabe führt damit zu mehr Bearbeitungszuständen bei Aktivitäten.

- Im Vergleich zu Ausführungsmodellen anderer Workflow-Management-Systeme muss zur Unterstützung vorzeitiger Datenweitergabe auch die übliche Schaltlogik von Kontrollflussknoten (bedingte und parallele Verzweigungen, Schleifen) verändert werden. Einem Kontrollflussknoten werden zusätzliche Zustandsübergänge von Aktivitäten signalisiert, die er geeignet verarbeiten und an nachfolgende Aktivitäten zum richtigen Zeitpunkt weiterleiten muss.

- Zur Konsistenzsicherung der Daten ist bei einem Modell mit vorzeitiger Datenweitergabe auch die Verwaltung von *Objektversionen* unabdingbar. Für verschiedene Aktivitäten können damit zu einem Zeitpunkt unterschiedliche Objektversionen des gleichen globalen Objekts gültig sein. Daraus resultieren zum einen mehr Bearbeitungszustände bei den Objektversionen. Zum anderen können sich die Bearbeitungszustände nicht auf ein globales Objekt als Ganzes, sondern müssen sich auf jede Objektversion eines globalen Objekts beziehen.

Es ist damit offensichtlich, dass sich das **WEP**-Ausführungsmodell aufgrund der umfangreicheren **WEP**-Benutzerinteraktionen und der daraus resultierenden größeren Anzahl von Bearbeitungszuständen sowie der komplexeren internen Schaltlogik erheblich von Ausführungsmodellen anderer Workflow-Management-Systeme unterscheidet. Das **WEP**-Ausführungsmodell wird nun im Detail vorgestellt.

Dazu wird zuerst als Verallgemeinerung der im Kapitel 11 exemplarisch skizzierten Benutzerinteraktionsfolgen das Schema *aller* erlaubten **WEP**-Benutzerinteraktionsfolgen definiert (Abschnitt 13.1). Nach Einführung einer geeigneten Darstellungsform für Workflow-Instanzen in Abschnitt 13.2 gibt Abschnitt 13.3 einen Überblick über alle Bearbeitungszustände von Aktivitäten und Objektversionen, um dann festzulegen, welche Benutzerinteraktionen in welchen Bearbeitungszuständen ausgeführt werden können, und welche Ereignisse eine Benutzerinteraktion an Kontroll- und Datenflusskanten übermittelt. (Abschnitt 13.4 und zusammenfassend in Abschnitt 13.7). Der Abschnitt 13.5 präzisiert das Schaltverhalten von Kontrollflussknoten. Als Ergebnis dieses Kapitels wird jeder Zustandsübergang von Objektversionen (Abschnitt 13.6) und Aktivitäten (Abschnitt 13.8) mit einer präzisen Übergangsbedingung belegt sein. Sie bilden damit die Grundlage für den Nachweis dynamischer Eigenschaften von **WEP**-Workflows in Kapitel 14.

## 13.1 Schema der erlaubten WEP-Benutzerinteraktionen

Abbildung 13.1 zeigt als Verallgemeinerung der im vorherigen Kapitel skizzierten Benutzerinteraktionen das Schema für alle erlaubten und zustandsverändernden Interaktionsfolgen im **WEP**-Workflow-Management-System in Form eines UML-Aktivitätsdiagramms.[1] Charakteristisch für erlaubte Interaktionsfolgen ist hierbei zum einen die Trennung zwischen dem Starten einer Aktivität ($StartActivity$) und der Entgegennahme ihrer Parameter ($FetchObjectVersion$) und zum anderen die Trennung zwischen Beenden der Aktivität ($FinishActivity$) und der Weitergabe ihrer Ergebnisse ($ReleaseObjectVersion$).

Beide Charakteristika spiegeln die Forderung nach maximaler Flexibilität und nach Simultaneous-Engineering entlang der Prozesskette wider. Sie ermöglichen die Integration neu eingetroffener Objektversionen und die frühzeitige Weitergabe bereits vorliegender Ergebnisse noch während der Aktivitätenbearbeitung.

Innerhalb der Klammer der Benutzerinteraktionen $StartActivity$ und $FinishActivity$ können die Operationen $FetchObjectVersion$ und $CreateObjectVersion$ beliebig häufig durchgeführt werden, um somit Bearbeitern einer Aktivität die Möglichkeit zu geben, gegebenenfalls verschiedene Varianten von Arbeitsergebnissen erzeugen, diese zu vergleichen und schrittweise von einer Qualitätsstufe zur geforderten nächsthöheren Qualitätsstufe zu gelangen.

[1] Aus Übersichtsgründen wurden die für die folgenden formalen Betrachtungen nicht relevanten Benutzerinteraktionen, wie beispielsweise das Starten eines Subprozessschrittes oder das Suspendieren einer Aktivität nicht dargestellt.

Es gilt allerdings die Einschränkung, dass die Interaktionen $CreateObjectVersion$ nur auf Objektversionen im Arbeitsbereich und $FetchObjectVersion$ nur auf Objektversionen aus dem Eingabe- oder Ausgabebereich angewandt werden dürfen. Es muss also zumindest beim ersten Ableiten einer Objektversion die Operation $FetchObjectVersion$ durchgeführt werden, damit sich eine Objektversion im Arbeitsbereich befindet. Darüber hinaus sind nur Interaktionsfolgen erlaubt, bei denen jedem $ReleaseObjectVersion$ mindestens eine $FetchObjectVersion$- oder $CreateObjectVersion$-Operation vorangeht. Dies garantiert, dass jede Objektversion höchstens einmal an Folgeschritte weitergegeben wird.

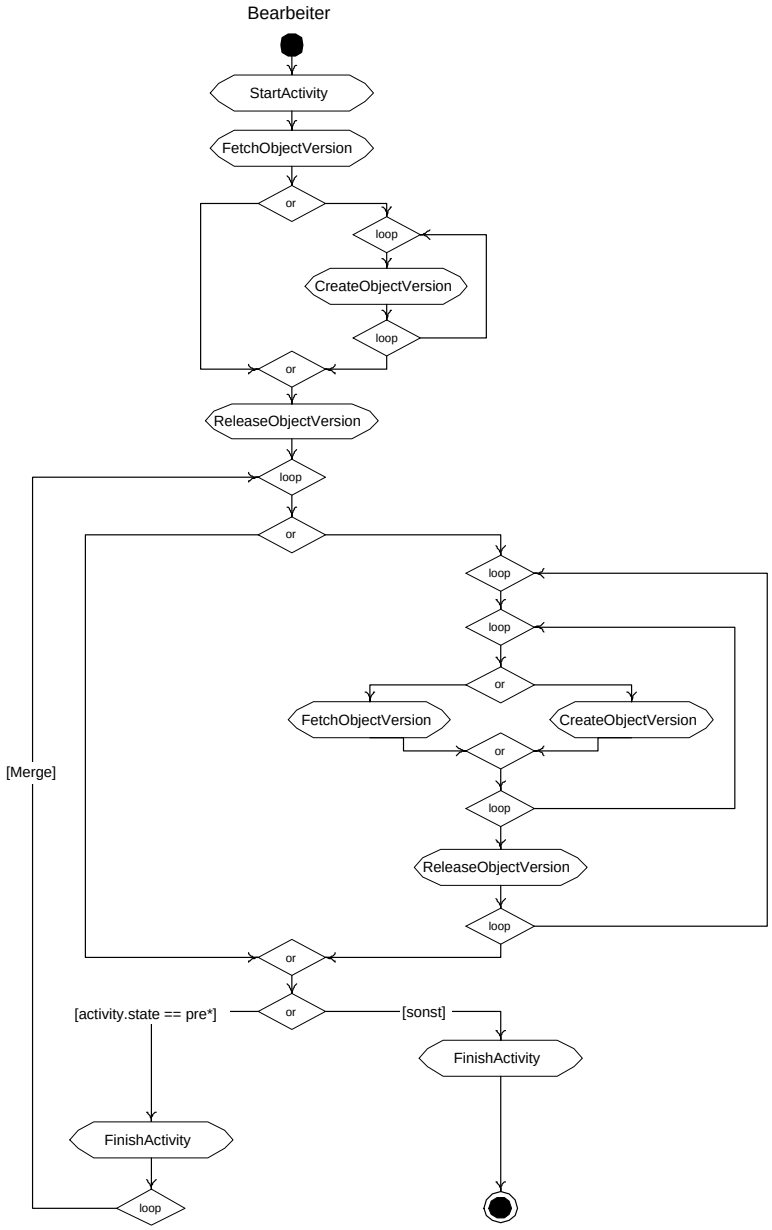



**Abbildung 13.1:** Schema aller erlaubten und zustandsverändernden Benutzerinteraktionsfolgen

## 13.2 Erweiterung des Workflow-Graphen um Laufzeitaspekte

Zur Steuerung eines **WEP**-Workflows wird die im Teil II eingeführte graphbasierte Workflow-Beschreibung eines **WEP**-Workflow-Schemas herangezogen und um Laufzeitinformationen ergänzt. Diese Laufzeitinformationen beinhalten im Wesentlichen:

- die Bearbeitungszustände bei Aktivitäten und Objektversionen
- die Ereignisse auf den Kontroll- und Datenflusskanten

Das eine Aktivität beschreibende Tupel (vergleiche Abschnitt 6.2) wird um die folgenden Elemente ergänzt:

$Astate$:
: $Astate \in AstateSet$ beschreibt den aktuellen Bearbeitungszustand der Aktivität. Die Menge $AstateSet$ enthält die folgenden Elemente: $notOffered$, $offered$, $preOffered$, $activated$, $preActivated$, $preConsRecov$, $finished$ und $preFinished$, deren Bedeutung in Abschnitt 13.3 dargelegt wird.

$AactivationCounter \in \mathbb{N}$:
: Der Aktivierungszähler $AactivationCounter$ dokumentiert die Anzahl der Aktivierungen einer Aktivität. Eine im Workflow-Schema beschriebene Aktivität kann aufgrund von Schleifen mehrmals aufgerufen werden.

Eine Objektversion wird zur Laufzeit mittels folgender Merkmale spezifiziert:

$OVglobalObject$:
: $OVglobalObject \in GLOBALOBJECTS$ referenziert auf das globale Objekt, zu dem die Objektversion gehört.

$OVclass$:
: Dieses Attribut leitet sich aus der Klasse des globalen Objekts $O$, zu dem die Objektversion gehört, ab. Es gilt also: $OVclass$ = $GOclass(O)$.

$OVstate$:
: Mittels $OVstate \in OVstateSet = \{unset, locked, preReleased, released\}$ wird der aktuelle Zustand einer Objektversion spezifiziert (vergleiche auch Abbildung 13.3).

$OVcreator$:
: Hiermit wird die erzeugende Aktivität einer Objektversion dokumentiert.

Um Zustandsveränderungen mitzuteilen oder zu veranlassen, werden Ereignisse über Kontroll- und Datenflusskanten an andere Knoten im **WEP**-Workflow-Graph weitergeleitet, die diese verarbeiten und eventuell weitere Ereignisse auslösen. Manche Ereignisse können dabei mit dem Returncode der auslösenden Aktivität parametrisiert sein. Zur Darstellung der Ereignisse wird die Beschreibung von Kontroll- und Datenflusskanten (siehe Abschnitte 9.3.1 und 8.2) um das Attribut $CFevent$ beziehungsweise $DFevent$ ergänzt.

Ereignisse werden im **WEP**-Ausführungsmodell nach folgenden Regeln verarbeitet. Bei der Verarbeitung eines Ereignisses durch einen Knoten wird es „verbraucht“. Es wird also von der Kante gelöscht. Ereignisse müssen dabei nicht sofort verarbeitet werden. Sie können auf der Kante verbleiben, bis weitere Ereignisse über andere Knoten eintreffen, die zum Schalten des Knotens notwendig sind.

Trifft ein weiteres Ereignis über dieselbe Kante ein, so wird ein eventuell noch vorhandenes Ereignis überschrieben.

## 13.3 Bearbeitungszustände von Aktivitäten und Objektversionen

Die Abbildung 13.2 zeigt zusammenfassend die möglichen Bearbeitungszustände[2] von Aktivitäten sowie alle erlaubten Zustandsübergänge in Form eines endlichen Automaten. Die Bedingungen für einen Zustandübergang werden im weiteren Verlauf des Kapitel 13 spezifiziert.

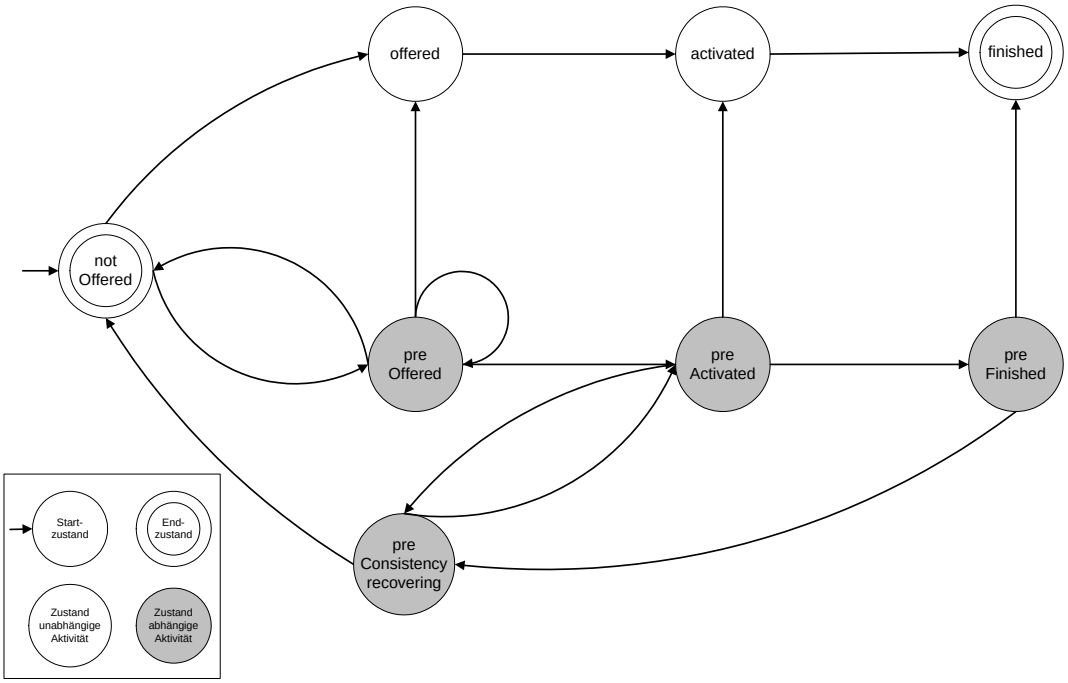



**Abbildung 13.2:** Vereinfachter Überblick über die erlaubten Zustandsübergänge bei Aktivitäten

Die Aktivitätenzustände lassen sich grob in zwei Gruppen aufgliedern. Die eine Gruppe repräsentiert Bearbeitungszustände abhängiger Aktivitäten. Diese Zustände wurden mit dem Präfix $pre$ gekennzeichnet. Die andere Gruppe enthält die Zustände unabhängiger Aktivitäten. Die Bearbeitungszustände bedeuten im Einzelnen:

*$notOffered$:*
: Dieser Zustand $notOffered$ repräsentiert den Initialzustand einer Aktivität. Er wird von einer Aktivität ebenfalls wieder (als Endzustand) eingenommen, wenn sie aufgrund aktueller Daten zurückgesetzt wird oder wenn ein anderer Pfad gewählt wurde, bei dem die Bearbeitung dieser Aktivität nicht notwendig ist.

*$offered$/$preOffered$:*
: Der Zustand $offered$ beziehungsweise $preOffered$ wird von einer unabhängigen beziehungsweise abhängigen Aktivität eingenommen, wenn sie zur Bearbeitung angeboten wird,

[2] Die gezeigten Bearbeitungszustände stellen einen Auszug aller Bearbeitungszustände im **WEP**-Workflow-Management-System dar. Aus Gründen der Übersichtlichkeit wurden nur diejenigen dargestellt, die für die folgenden Überlegungen von Belang sind.

also in den Arbeitslisten potenzieller Bearbeiter eingefügt ist, aber noch von keinem der Bearbeiter bearbeitet wird. Aktivitäten in diesen Zuständen nennt man auch *aktivierbar*.

$activated/preActivated$:
Diese Zustände repräsentieren eine unabhängige beziehungsweise abhängige Aktivität in Bearbeitung.

$finished/preFinished$:
Eine unabhängige beziehungsweise abhängige Aktivität wurde beendet.

$preConsRecov$:
Während einer Konsistenzsicherungsphase befindet sich eine Aktivität im Zustand $preConsRecov$.

Objektversionen können potenziell die in Abbildung 13.3 dargestellten Bearbeitungszustände einnehmen.

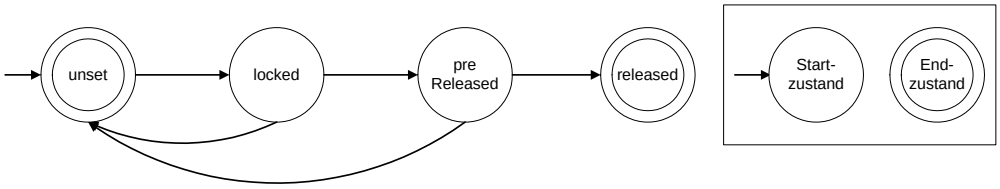



**Abbildung 13.3:** Vereinfachter Überblick über die erlaubten Zustandsübergänge bei Objektversionen

Auch hier werden die Zustandübergangsbedingungen noch innerhalb des Kapitels 13 definiert. Die einzelnen Zustände bedeuten:

$unset$: Initialzustand für noch nicht erzeugte Objektversionen beziehungsweise Endzustand ungültiger Versionen.

$locked$:
Zustand einer Objektversion im Arbeitsbereich einer Aktivität.

$preReleased$:
Zustand einer Objektversion, die von einer noch nicht beendeten Aktivität (als vorläufiges Datum) weitergegeben wurde.

$released$:
Zustand einer Version, wenn die erzeugende Aktivität endgültig beendet wurde.

## 13.4 Formales Systemverhalten bei den WEP-Benutzerinteraktionen

Dieser Abschnitt beschreibt, wie Bearbeiter und das **WEP**-Laufzeitsystem miteinander agieren. Dazu wird für jede Benutzerinteraktion aus Abbildung 13.1 in Form von UML-Aktivitätsdiagrammen spezifiziert, welche Voraussetzungen für die Durchführung einer Interaktion vorherrschen müssen, welche Zustandsänderungen sie bewirken und welche Ereignisse sie über Daten- und Kontrollflusskanten auslösen.

### 13.4.1 Formales Systemverhalten bei der Interaktion $StartActivity$

Wie aus dem UML-Aktivitätsdiagramm aus Abbildung 13.4 ersichtlich ist, kann die Benutzerinteraktion $StartActivity$ nur auf Aktivitäten angewandt werden, die sich im Bearbeitungszustand $preOffered$ beziehungsweise $offered$ befindet. Ist sie eine abhängige Aktivität (Bearbeitungszustand $preOffered$), so verändert sich ihr Zustand zu $preActivated$. Andernfalls befindet sie sich im Bearbeitungszustand $activated$. In beiden Fällen wird das Ereignis $evActivated$ an der Kontrollflussausgangskante ausgelöst. Dieses Ereignis signalisiert nachfolgenden Aktivitäten, dass ihre Vorgängeraktivität gestartet wurde. Aus Sicht des Kontrollflusses ist damit eine Nachfolgeraktivität als abhängige Aktivität aktivierbar.

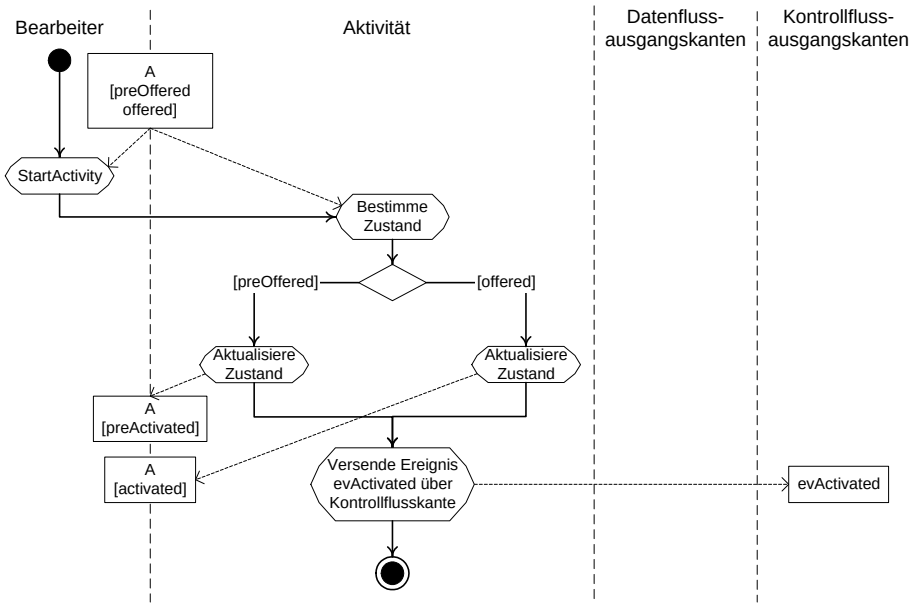



**Abbildung 13.4:** Systemverhalten bei der Benutzerinteraktion $StartActivity$

### 13.4.2 Formales Systemverhalten bei der Interaktion $FinishActivity$

Durch die Interaktion $FinishActivity$ beendet ein Bearbeiter eine Aktivität. Ihr Zustand ändert sich damit von $preActivated$ beziehungsweise $activated$ zu $preFinished$ oder $finished$ (siehe Abbildung 13.5).

Voraussetzung für die Durchführung dieser Operation ist, dass mindestens für einen Returncode alle zwingend erforderlichen Meilensteine erreicht worden sind (vergleiche Abschnitt 6.2.2.3). Das Ende einer Aktivität wird an nachfolgende Aktivitäten über das Ereignis $evFinished(rc)$ bei einer unabhängigen Aktivität beziehungsweise über das Ereignis $evPreFinished(rc)$ bei einer abhängigen Aktivität mitgeteilt. Beide Ereignisse enthalten den gewählten Returncode $rc$ der beendeten Aktivität. Im Falle einer unabhängigen Aktivität (neuer Bearbeitungszustand $finished$) wird zusätzlich über ihre Datenflussausgangskanten mittels dem Ereignis $evObjectVersionReleased$ signalisiert, dass sie keine weiteren Objektversionen erzeugen wird. Objektversionen, die mit Arbeitsparametern der Aktivität verbunden sind oder mit Ausgabeparametern verknüpft sind, die für den gewählten Returncode nicht relevant sind, werden von einer unabhängigen (abhängigen) Aktivität mittels dem Er-

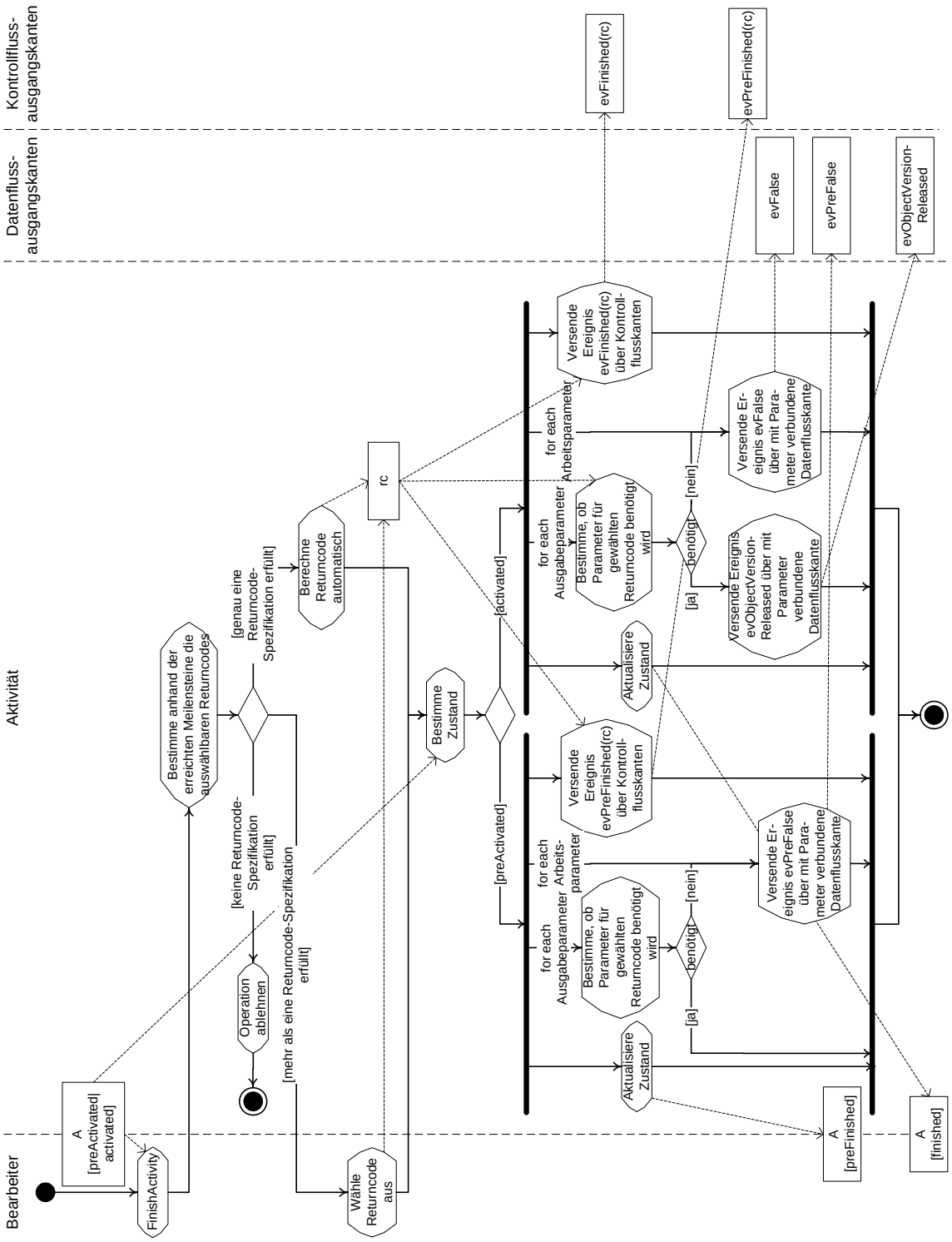



**Abbildung 13.5:** Systemverhalten bei der Benutzerinteraktion $FinishActivity$

eignis $evFalse$ ($evPreFalse$) informiert, dass sie nicht mehr benötigt werden. Die Differenzierung zwischen den hier gezeigten $pre$- und nicht-$pre$-Datenflussereignissen ermöglicht dabei dem empfangenden globalen Objekt zu entscheiden, ob es den lesenden Aktivitäten das Signal für endgültige Eingabedaten ($evFinalObjectVersion$) senden kann.

### 13.4.3 Formales Systemverhalten bei den Interaktionen $FetchObjectVersion$ und $CreateObjectVersion$

Mittels der Benutzerinteraktion $FetchObjectVersion$ kann ein Aktivitätenbearbeiter von einer bereits vorhandenen Objektversion[3], die er durch Auswahl des mit ihr verbundenen Ein- oder Ausgabeparameter bestimmt, eine weitere Objektversion ableiten. Auf diese Objektversion, die nur für ihn sichtbar und veränderbar ist, kann er über einen ebenfalls erzeugten, im Arbeitsbereich der Aktivität liegenden Arbeitsparameter zugreifen (siehe Abbildung 13.6). Die Beauftragung zum Erzeugen einer neuen Objektversion, der entsprechenden Datenflusskanten, des Arbeitsparameters und der aktuellen Eingabeparameter für die lesenden Aktivitäten erfolgt über das Ereignis $evNewObjectVersionRequested$.

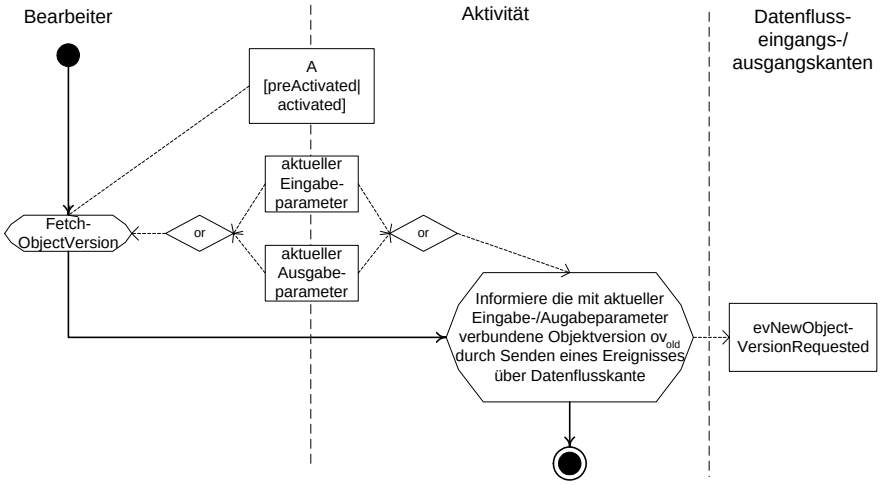



**Abbildung 13.6:** Systemverhalten bei der Benutzerinteraktion $FetchObjectVersion$

Die Interaktion $CreateObjectVersion$ weist das gleiche Systemverhalten auf wie $FetchObjectVersion$. Sie kann allerdings nur auf Arbeitsparameter angewendet werden (siehe Abbildung 13.7).

### 13.4.4 Formales Systemverhalten bei der Interaktion $ReleaseObjectVersion$

Während Bearbeiter „innerhalb“ ihrer Aktivitäten viel Freiheiten besitzen, müssen Objektversionen trotz ihres im Allgemeinen vorläufigen Charakters gewissen „Qualitätsstandards“ genügen. Diese Standards wurden bei der Modellierung einer zielorientierten Aktivität (vergleiche Abschnitt 6.2.2.2)

[3] Während der Workflow-Initialisierung erhält jedes globale Objekt zumindest eine „leere“ Wurzel-Objektversion (siehe Kapitel 11).

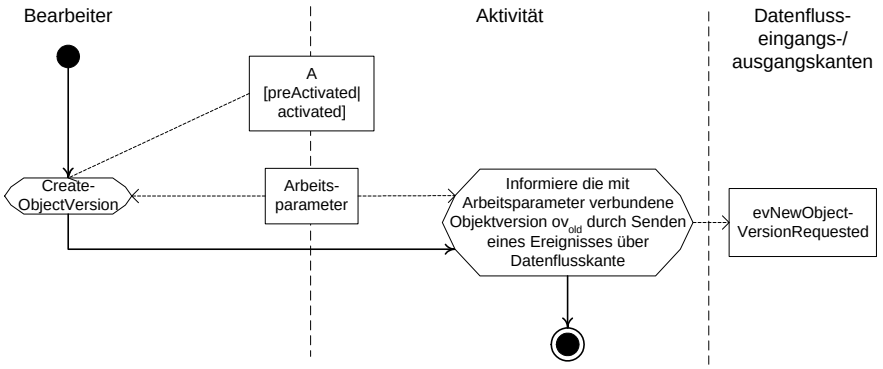



**Abbildung 13.7:** Systemverhalten bei der Benutzerinteraktion $CreateObjectVersion$

in Form von Meilensteinspezifikationen festgelegt. Genügt eine Objektversion keiner Meilensteinspezifikation, so ist eine Weitergabe nutzlos, da keine der im Datenfluss nachfolgenden Aktivitäten diese benötigt. Die in Abschnitt 9.3.6 spezifizierten Datenflussregeln erlauben aus diesem Grund nur dann das Ziehen von Datenflusskanten zwischen erzeugenden und lesenden Aktivitäten, wenn diese auch kompatibel zu den spezifizierten Qualitätsstufen sind (siehe Definition 8 von *korrekten* beziehungsweise *SE-optimierten* Datenflusskanten). Die Einhaltung von mindestens einer Meilensteinspezifikation wird deshalb bei der Durchführung der Interaktion $ReleaseObjectVersion$ überprüft. Wird keine Übereinstimmung gefunden, wird die Operation abgelehnt. Ansonsten wird der Weitergabewunsch an das globale Objekt über das Ereignis $evObjectVersionPreReleased$ signalisiert. Weitere Details der Benutzerinteraktion sind im UML-Aktivitätsdiagramm der Abbildung 13.8 zu finden.

## 13.5 Schaltlogik der Kontrollflussknoten

Neben zielorientierten Aktivitäten, globalen Objekten als Container der Objektversionen, Kontroll- und Datenflusskanten enthält der **WEP**-Workflow-Graph noch Knoten, welche die Kontrollflussterminale für bedingte und parallele Verzweigungen sowie für Schleifen repräsentieren. Durch die Repräsentation der Kontrollflussterminale als eigene Knoten im Graph, sogenannte Kontrollflussknoten, reduzieren sich die Fälle bei der formalen Beschreibung der Vorbedingungen für Aktivitätenzustände, da jede Aktivität genau eine Kontrollflusseingangskante und -ausgangskante besitzt. Wann die Kontrollflusseingangskante einer Aktivität schaltet, hängt von der Schaltsemantik des vorangehenden Kontrollflussknotens ab. Kontrollflussknoten besitzen keinen eigenen Bearbeitungszustand. Sie enthalten jedoch eine Schaltlogik, die im Folgenden spezifiziert wird.

Ein Kontrollflussknoten besitzt wie Aktivitätenknoten Kontrollflusseingangs- und -ausgangskanten, über die Ereignisse gesendet werden. Es existieren vier Typen von Kontrollflussknoten, die aufgrund der strikten Blockstruktur immer paarweise auftreten (vergleiche Abschnitt 9.3.1). Abhängig vom eintreffenden Ereignis und der Schaltsemantik des Kontrollflussknotens werden über die Kontrollflussausgangskanten verschiedene Ereignisse weitergeleitet. Dieses Schaltverhalten wird nun spezifiziert. Für die im Folgenden benötigten Überlegungen genügt dabei eine anschauliche Beschreibung der Schaltsemantik aller **WEP**-Kontrollflussknoten. Auf eine detaillierte Beschreibung wird deshalb an dieser Stelle verzichtet. Der interessierte Leser findet diese in Form weiterer UML-Aktivitätsdiagramme im Anhang B.2.

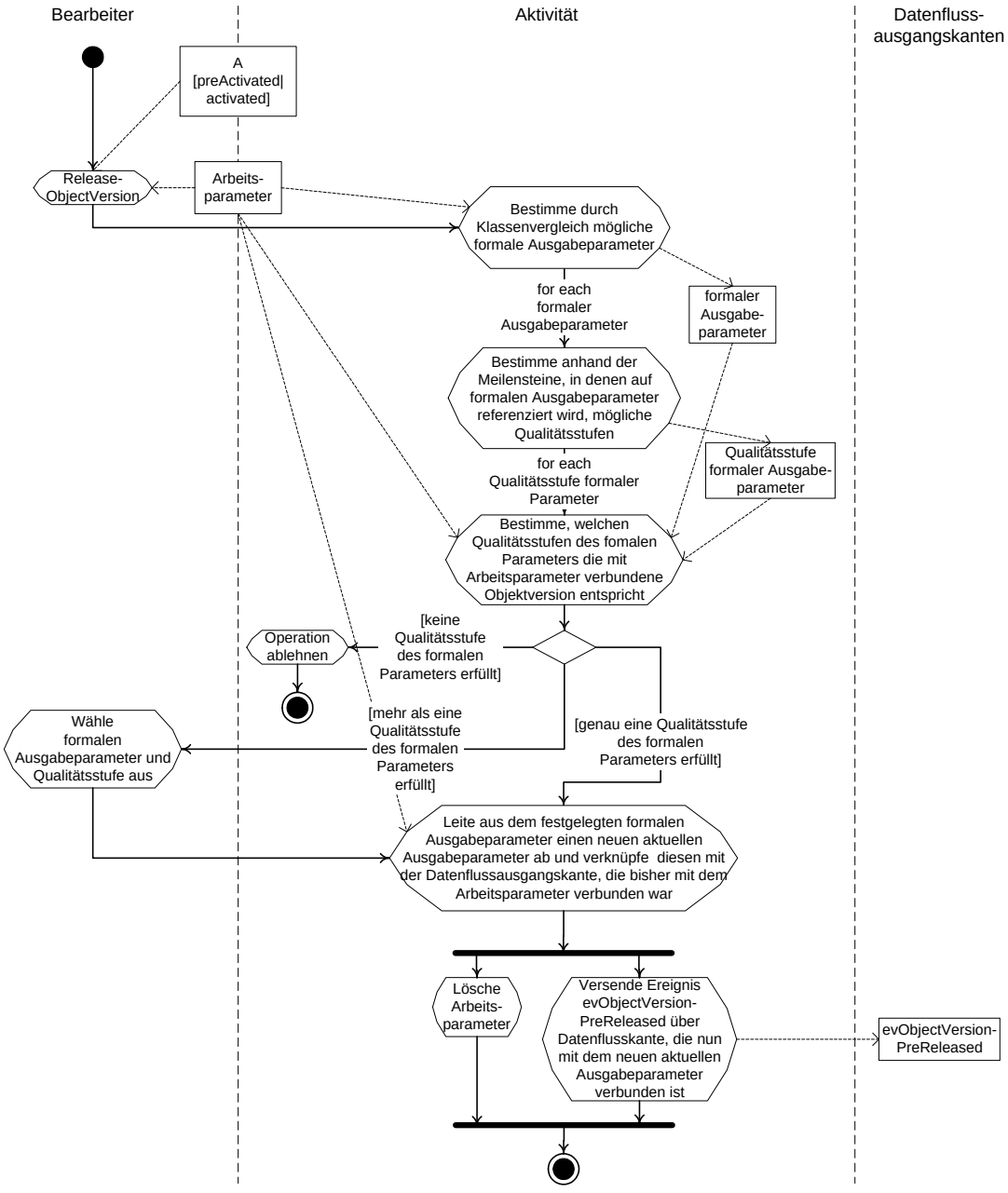



**Abbildung 13.8:** Systemverhalten bei der Benutzerinteraktion $ReleaseObjectVersion$

### 13.5.1 Parallele Verzweigung

Bei der Betrachtung der Schaltlogik der Kontrollflussknoten muss nicht zwischen dynamischer und statischer Parallelität unterschieden werden, da die dynamische Aufsplittung des Kontrollflusses bereits vor dem Eintreffen der ersten Ereignisse beim Kontrollflussknoten, der den Beginn einer dynamischen Parallelität markiert, erfolgt ist.

Abbildung 13.9 zeigt die Darstellung einer parallelen Verzweigung mit expliziten Kontrollflussknoten. Der den Beginn einer parallelen Verzweigung symbolisierende Kontrollflussknoten wird schalten, sobald die Aktivität $A1$ gestartet wurde, der Kontrollflussknoten also das Ereignis $evActivated$ empfangen hat. Aktivitäten $A2$, ..., $An$ sind damit aus Kontrollflusssicht als abhängige Aktvitäten aktivierbar.

Der das Ende einer parallelen Verzweigung symbolisierende Kontrollflussknoten wird erst dann eintreffende Ereignisse weiterleiten, wenn *alle* Aktivitäten $A2$, ..., $An$ aktiviert wurden. Das erste Kontrollflussereignis, das Aktivität $An{+}1$ erhalten wird, ist also immer $evActivated$.

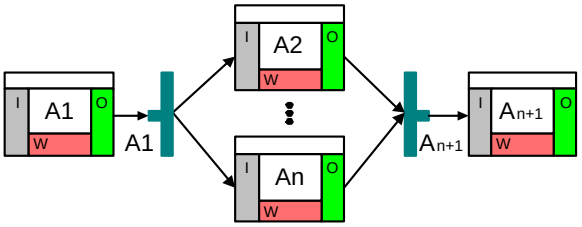



**Abbildung 13.9:** Parallele Verzweigung mit expliziten Kontrollflussknoten

Beim Kontrollflussknoten können neben dem Ereignis $evActivated$ auch noch die Ereignisse $evFinished(rc)$ beziehungsweise $evPreFinished(rc)$ (endgültiges beziehungsweise vorläufiges Beenden der jeweiligen Vorgängeraktivität), $evTrue$ (endgültiges Verlassen einer Schleife, siehe Abschnitt 13.5.3) oder $evFalse$ beziehungsweise $evPreFalse$ (erneuter Schleifendurchlauf, ausgelöst durch unabhängige beziehungsweise abhängige Aktivität) eintreffen.

Bei den Ereignissen $evFinished(rc)$ und $evTrue$ verfährt der Kontrollflussknoten analog zum Ereignis $evActivated$ nach der (n-of-n)-Schaltlogik. Er leitet also in beiden Fällen erst dann ein Ereignis, nämlich $evTrue$, weiter, wenn er von *allen* Vorgängeraktivitäten das Ereignis $evFinished(rc)$ oder $evTrue$ erhalten hat. Er signalisiert dadurch seiner nachfolgenden Aktivität, dass alle Vorgängeraktivitäten endgültig beendet sind.

Das Ereignis $evPreFinished(rc)$, welches das vorläufige Beenden einer abhängigen Aktivität signalisiert, wird dagegen nicht weitergeleitet, da es für die nachfolgenden Aktivitäten irrelevant ist: Als Vorbedingung zum Anbieten der Nachfolgeraktivität als abhängige Aktivität genügt der Start der Vorgängeraktivitäten (signalisiert durch Ereignis $evActivated$). Der Übergang zur Unabhängigkeit kann erst erfolgen, wenn die Vorgängeraktivitäten endgültig (Ereignis $evFinished(rc)$ oder $evTrue$) und nicht nur vorläufig beendet sind.

Eintreffende Ereignisse vom Typ $evFalse$ beziehungsweise $evPreFalse$ werden dagegen sofort weitergeleitet. Beide Ereignisse signalisieren, dass durch Bearbeiten der Aktivitäten $A_{after}$ „hinter“ dem Kontrollflussknoten (vorläufig) der falsche Weg durch den Workflow-Graph beschritten wird, da mindestens bei einem parallelen Zweig vor dem Knoten eine Schleife existiert, die erneut durchlau-

fen werden muss. Zur Vermeidung der Fehlerfortpflanzung oder zumindest von unnötiger Arbeit müssen die Aktivitäten $A_{after}$ sofort benachrichtigt werden, wenn ein $evFalse$- oder $evPreFalse$-Ereignis eintrifft. Hier ist deshalb für ein Kontrollflussknoten das (1-of-n)-Schaltverhalten adäquat. Abbildung 13.10 veranschaulicht diesen Sachverhalt.

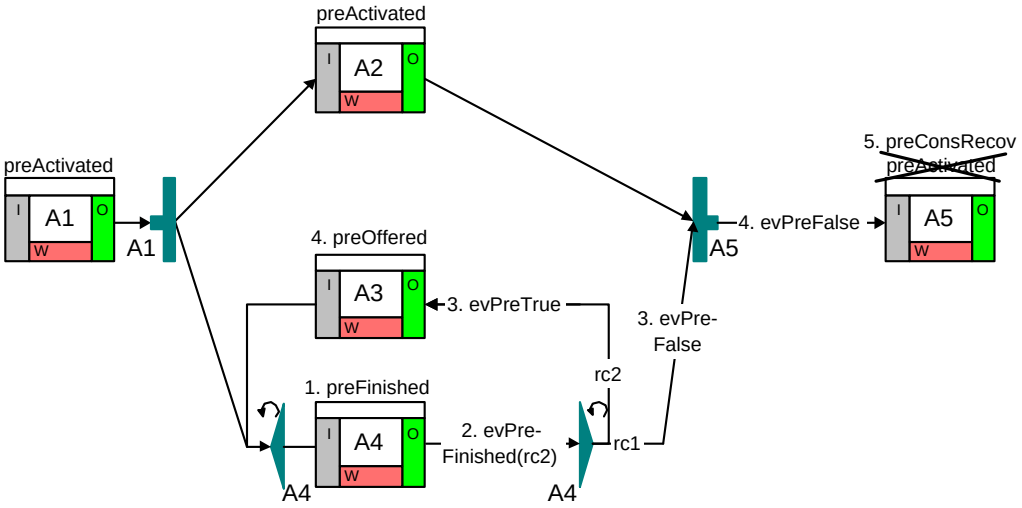



**Abbildung 13.10:** Schaltverhalten eines Kontrollflussknotens beim Eintreffen eines $evFalse$/$evPreFalse$-Ereignisses. Aktivität $A4$ löst einen neuen Schleifendurchlauf aus. Der bereits vorläufig beschrittene Kontrollflussweg von $_{A4}$ zu $_{A5}$ wird damit ungültig.

Da die Kontrollflussknoten und – wie auch alle anderen **WEP**-Kontrollflussknoten – bereits beim Start der assoziierten Aktivität schalten, unterscheidet sich die **WEP**-Schaltlogik von der Ausführungssemantik anderer Workflow-Management-Systeme, bei denen Kontrollflussknoten erst mit Beendigung der assoziierten Aktivitäten schalten.

Diese **WEP**-Schaltsemantik ist notwendig, um Simultaneous-Engineering-Phasen auch über Kontrollflussknoten hinweg einläuten zu können. Wie später ersichtlich wird, schalten alle **WEP**-Kontrollflussknoten auch beim Aktivitätenende. Sie schalten damit im Gegensatz zu anderen Ausführungsmodellen auch ohne Vorhandensein von Schleifen mehrmals.

### 13.5.2 Bedingte Verzweigung

Die Abbildung 13.11 repräsentiert das Basiskonstrukt einer bedingten Verzweigung mit expliziten Kontrollflussknoten.

Auch hier ist zur Unterstützung von Simultaneous-Engineering ein Aktivieren von Aktivität $A1$ nachfolgenden Aktivitäten $A2, \ldots, An$ bereits nach Start von $A1$ unabdingbar. Im Gegensatz zu einer parallelen Verzweigung wird jedoch nur *einer* der $n$ bedingten Zweige am Ende durchlaufen werden. Es stellt sich also die Frage, welche der bedingten Zweige beim Start der Aktivität $A1$ aktivierbar gesetzt werden sollen. Da der „korrekte“ Zweig jedoch erst beim Beenden der Aktivität $A1$ durch Festlegung des Returncodes feststeht, steht diese Information beim ersten Schalten des Kontrollflussknotens nicht zur Verfügung.

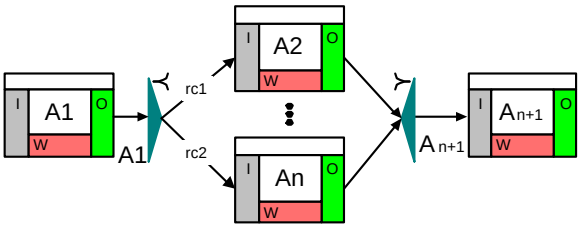



**Abbildung 13.11:** Bedingte Verzweigung mit expliziten Kontrollflussknoten

Um minimale Prozessdurchlaufzeiten zu erzielen, werden beim **WEP**-Ausführungsmodell deshalb immer alle bedingten Zweige aktivierbar gesetzt. Ein Kontrollflussknoten $\blacktriangleright^{\prec}$ zeigt also durch den Start der Vorgängeraktivität das gleiche Schaltverhalten wie ein paralleler Verzweigungsknoten $\blacksquare$.

Anders verhält es sich, wenn der Returncode der Vorgängeraktivität feststeht. Wird dies einem Kontrollflussknoten $\blacktriangleright^{\prec}$ über das Ereignis $evPreFinished(rc)$ beziehungsweise $evFinished(rc)$ signalisiert, so schaltet dieser nur noch den mit den Returncode $rc$ markierten Zweig frei (Ereignis $evPreTrue$ beziehungsweise $evTrue$). Die anderen „falschen“ Zweige werden über die Ereignisse $evPreFalse$ beziehungsweise $evFalse$ zurückgesetzt (Undo-Konsistenzsicherung).

Am Ende einer bedingten Verzweigung wird genau dann eine Minimierung der Prozessdurchlaufzeiten erreicht, wenn der Knoten schaltet, sobald bei einem der $n$ eingehenden Kontrollflusskanten ein $evActivated$-Ereignis anliegt ((1-of-n)-Schaltverhalten). Die Aktivität $A_{n+1}$ in Abbildung 13.11 ist damit aus Kontrollflusssicht aktivierbar, wenn *eine* der Aktivitäten $A2, \ldots, An$ gestartet wurde.

Das gleiche (1-of-n)-Schaltverhalten wird angewandt, wenn die Ereignisse $evTrue$ oder $evFinished(rc)$ ankommen. Bei beiden Ereignissen leitet der Kontrollflussknoten $\succ\!\blacktriangleleft$ $evTrue$ weiter. Trifft das Ereignis $evPreFinished(rc)$ ein, so wird es ignoriert, da es für Nachfolgerschritte irrelevant ist.

Komplexer ist das Schaltverhalten am Ende einer bedingten Verzweigung, wenn eines der Ereignisse $evFalse$ und $evPreFalse$ eintrifft. Diese Ereignisse werden nur dann weitergeleitet, wenn die auslösende Aktivität in Bearbeitung ist beziehungsweise war und sie den Bearbeitungszustand der Aktivitäten $A_{after}{}_{\succ\!\blacktriangleleft}$ nach dem Kontrollflussknoten $\succ\!\blacktriangleleft$ verändern. Eine Veränderung aus Kontrollflusssicht tritt immer genau dann ein, wenn bisher nur einer der bedingten Zweige ein $evActivated$-Ereignis ausgelöst hat. Dies ist dann auch der Zweig, von dem das Ereignis $evFalse$ oder $evPreFalse$ eintrifft. Da bisher nur von einem Zweig die letzte Aktivität gestartet wurde und genau diese zurückgesetzt wird, ist die Voraussetzung für das Anbieten von Aktivitäten nach dem Kontrollflussknoten $\succ\!\blacktriangleleft$ nicht mehr vorhanden. Folgerichtig muss das Ereignis $evFalse$ oder $evPreFalse$ an sie weitergeleitet werden, damit auch bei ihnen die Undo-Konsistenzsicherung initiiert werden kann. Haben jedoch auch andere bedingte Zweige ein $evActivated$-Ereignis gesendet, dann besteht weiterhin die Voraussetzung für die Aktivitäten $A_{after}{}_{\succ\!\blacktriangleleft}$ aus Kontrollflusssicht. Das ankommende $evFalse$- oder $evPreFalse$-Ereignis muss vom Kontrollflussknoten nicht weitergeleitet werden. Das gewünschte Schaltverhalten kann mittels eines Zählers realisiert werden, der protokolliert, wie viele eingehende Kontrollflusszweige bereits das Ereignis $evActivated$ signalisiert haben (vergleiche Abbildung B.5).

Ein Zurücksetzen von Aktivitäten aus bedingten Zweigen muss auch betroffenen globalen Objekten über die Ereignisse $evFalse$ beziehungsweise $evPreFalse$ signalisiert werden, wenn diese Objektversionen besitzen, die von den zurückgesetzten Aktivitäten erzeugt wurden. Nur dadurch kann

bei dem soeben spezifizierten Schaltverhalten des Kontrollflussknotens $\succ\!\!\blacktriangleleft$ sichergestellt werden, dass alle Aktivitäten $A_{after}{}_{\succ\!\!\blacktriangleleft}$ so früh wie möglich informiert werden. Erhält ein globales Objekt ein $evFalse/evPreFalse$-Ereignis, so markiert es die betroffene Objektversion als ungültig und informiert die Aktivitäten, die auf diese Versionen zugegriffen haben. Die Abbildungen 13.12 und 13.13 veranschaulichen beide Sachverhalte.

### 13.5.3 Schleife

Die Schaltlogik einer Schleife, in ihrer allgemeinen Form in Abbildung 13.14 dargestellt, kann als eine bedingte Verzweigung betrachtet werden, deren einer Zweig den optionalen Schleifenrumpf enthält und deren anderer Zweig leer ist.

Die eine Schleife repräsentierenden Kontrollflussknoten $\blacktriangleright^{\circlearrowleft}$ und ${}^{\circlearrowleft}\!\blacktriangleleft$ weisen damit im **WEP**-Ausführungsmodell das gleiche Schaltverhalten wie die Kontrollflussknoten $\blacktriangleright^{\prec}$ beziehungsweise $\succ\!\!\blacktriangleleft$ der bedingten Verzweigung auf.

## 13.6 Schaltlogik der Objektversionen

Es fehlt nun noch das Schaltverhalten globaler Objekte beziehungsweise ihrer Objektversionen. Auch hier wird einer anschaulichen Darstellung der Vorzug vor detaillierten UML-Aktivitätsdiagrammen gegeben. Die Aktivitätsdiagramme sind im Anhang B.2 zu finden.

### 13.6.1 Erzeugung einer neuen Objektversion

Ein globales Objekt leitet eine neue Objektversion $ov_{new}$ von einer Version $ov_{old}$ ab, wenn über die Datenflusskante, die bei der Version $ov_{old}$ endet, das Ereignis $evNewObjectVersionRequested$ eintrifft. Die neu erzeugte Objektversion wird mit einem bei der auslösenden Aktivität neu erzeugten Arbeitsparameter über eine Datenflussausgangskante verbunden. Zusätzlich werden ein neuer aktueller Eingabeparameter und eine Datenflusseingangskante für jede von diesem globalen Objekt lesende Aktivität erzeugt.

### 13.6.2 Aktualisierung von Objektversionen

Eine Objektversion wird durch eine aktuellere ersetzt, wenn das Ereignis $evObjectVersionPreReleased$, ausgelöst durch die Benutzerinteraktion $ReleaseObjectVersion$, eintrifft. Die ältere Objektversion wechselt dabei in den Endzustand $unset$. Die aktuelle Version erhält den Zustand $preReleased$ und wird mit einem neuen ReleaseMarker markiert. Weitergegebene Objektversionen sind für andere Aktivitäten zugreifbar. Diese Aktivitäten werden über das Ereignis $evObjectVersionPreReleased$ informiert. Die Auswahl der zu benachrichtigenden Aktivitäten ist für beliebig strukturierte **WEP**-Workflow-Graphen komplex. Dieser Problematik widmet sich ausführlich Kapitel 15.

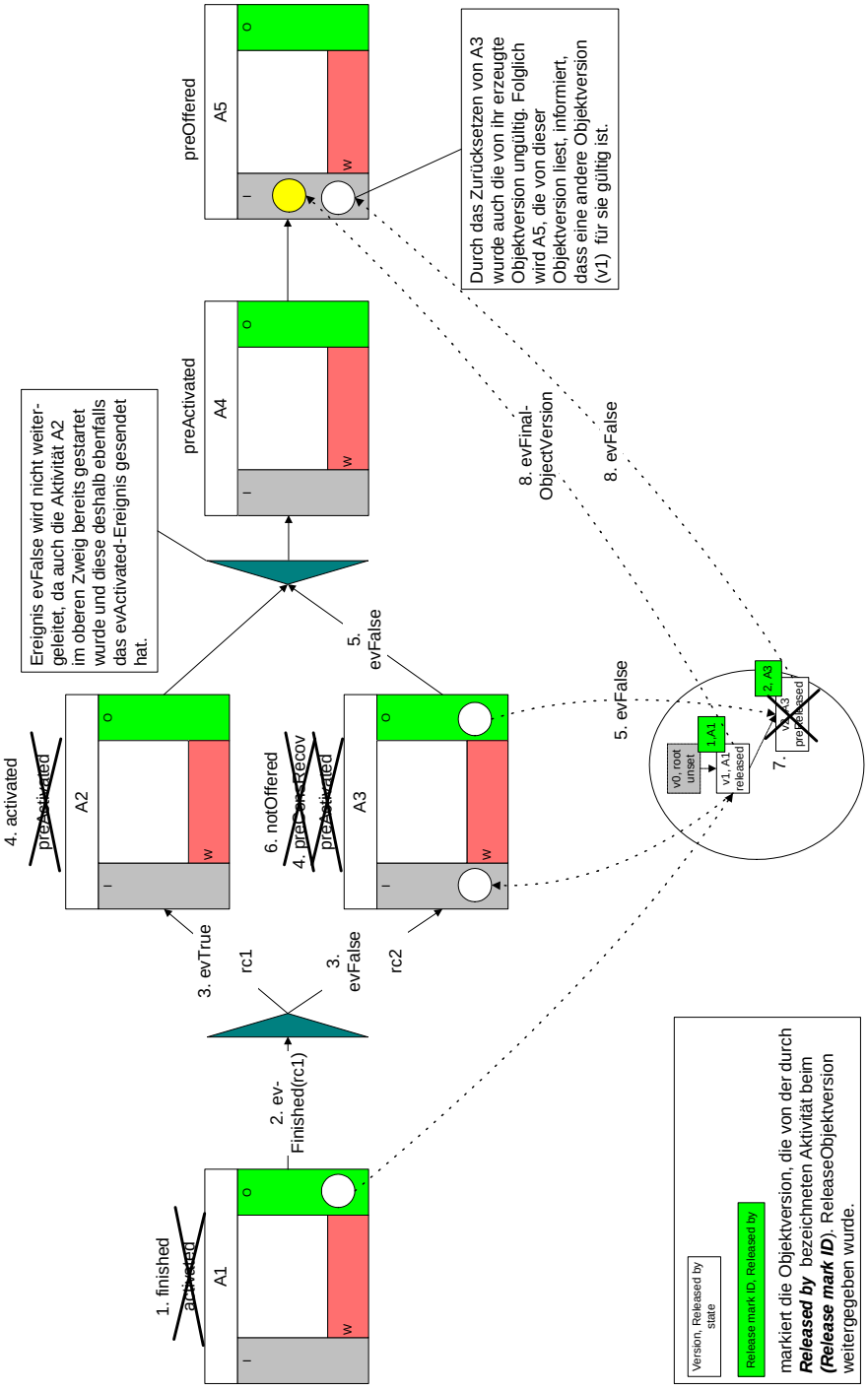



**Abbildung 13.12:** Schaltlogik des Kontrollflussknotens ⋋◀ beim $evFalse$-Ereignis, Beispiel 1

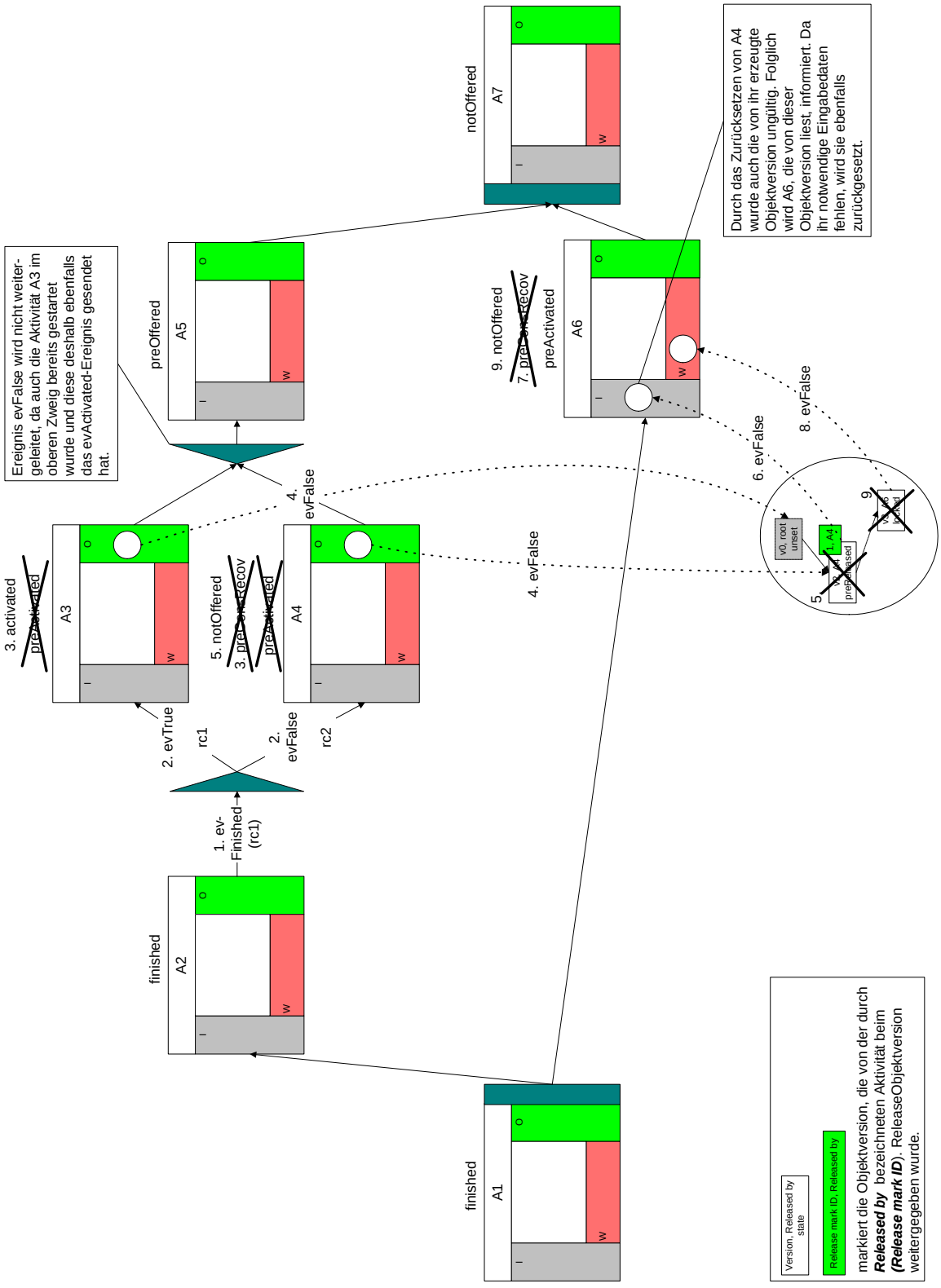



**Abbildung 13.13:** Schaltlogik des Kontrollflussknotens $\succ\!\!\blacktriangleleft$ beim $evFalse$-Ereignis, Beispiel 2

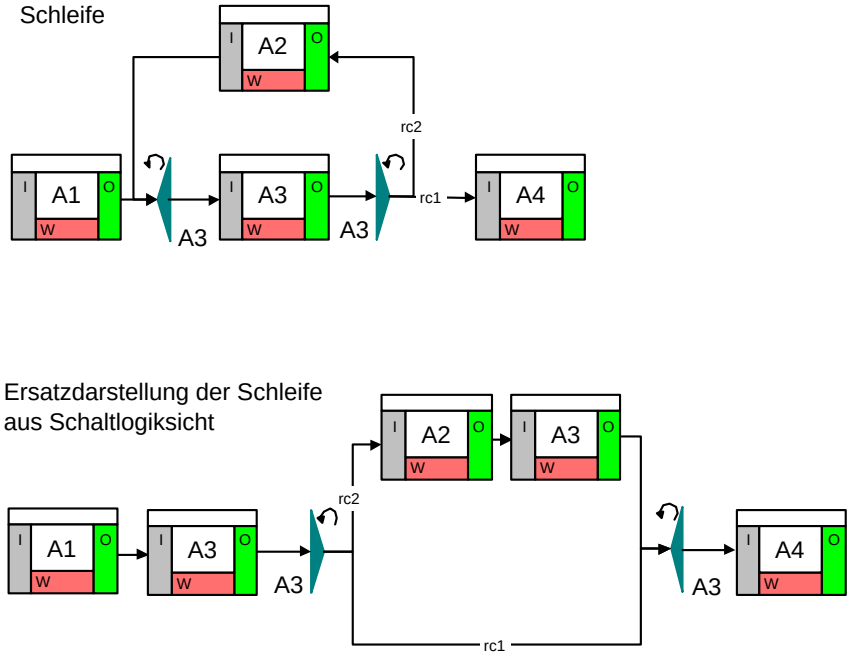



**Abbildung 13.14:** Schleife mit expliziten Kontrollflussknoten und ihre für das Schaltverhalten verwendete Ersatzdarstellung als bedingte Verzweigung

### 13.6.3 Endgültigkeit von Objektversionen

Die Benutzerinteraktion $FinishActivity$ löst das Ereignis $evObjectVersionReleased$ aus, um einem globalen Objekt zu signalisieren, dass die initiierende Aktivität keine neuen Objektversionen mehr erzeugen wird. Folgerichtig wechselt die Objektversion in den Endzustand $released$.

Eine Objektversion im Zustand $released$ repräsentiert nicht notwendigerweise gleich eine endgültige Objektversion für eine von diesem Objekt lesende Aktivität, da mehr als eine schreibende Aktivität existieren kann. Diese Konstellationen treten auf, wenn mehrere Kontrollflusspfade zu einer Aktivität existieren, bei denen jeweils eine schreibende Aktivität vorkommt. Aus diesem Grund werden für jeden Eingabeparameter $ip$ einer Aktivität $A$ die Menge $A.IPwriter^i(ip)$ verwaltet, die für jede Inkarnation[4] $i$ der betrachteten Aktivität alle schreibenden Aktivitäten enthält. Nur wenn alle Aktivitäten aus $A.IPwriter^i(ip)$ beendet sind – vermerkt in $TAS$ (*Terminated Activities Set*) – oder die von ihnen weitergegebenen Objektversionen nicht mehr an $A$ weitergeleitet werden (siehe Aktivitätenabhängigkeitsgraph in Kapitel 15), ist die zuletzt weitergegebene Objektversion im Zustand $released$ für eine lesende Aktivität endgültig. Das globale Objekt signalisiert dies der lesenden Aktivität mit dem Ereignis $evFinalObjectVersion$.

[4] Aufgrund von Schleifen kann eine bereits endgültig beendete Aktivität als neue Inkarnation nochmals angeboten und aktiviert werden. Eine neue Inkarnation wird durch Eintreffen des Ereignisses $evActivated$ ausgelöst.

### 13.6.4 Zurücksetzen von Objektversionen

Das Konzept der vorzeitigen Datenweitergabe und die Möglichkeit, Returncodes zu spezifizieren, bei denen nur ein Teil der Ausgabeparameter belegt sein müssen, führen dazu, dass bereits weitergegebene Objektversionen ungültig werden können. Eine ungültige Objektversion wechselt in den Endzustand $unset$. Soll eine Objektversion ungültig werden, so sendet die erzeugende Aktivität das Ereignis $evFalse$ (unabhängige Aktivität) beziehungsweise $evPreFalse$ (abhängige Aktivität).

Greifen andere Aktivitäten auf eine nun ungültige Objektversion zu, so müssen diese benachrichtigt werden. Hierbei können je nach Bearbeitungszustand des Workflows die folgenden Fälle auftreten:

1. Die ungültige Objektversion ist für lesende Aktivitäten nicht mehr relevant, da sie bereits durch eine aktuellere Version ersetzt wurde. Es muss dann noch überprüft werden, ob die aktuellere Objektversion durch das endgültige Beenden der die ungültige Objektversion erzeugenden Aktivität ein finales Eingabedatum für lesende Aktivitäten darstellt. Trifft dies für eine lesende Aktivität zu, so wird über die Datenflusskante, welche die nun endgültige Objektversion mit einem Eingabeparameter der Aktivität verbindet, das Ereignis $evFinalObjectVersion$ gesendet.

2. Die ungültige Objektversion kann durch eine frühere von einer anderen Aktivität erzeugten Objektversion ersetzt werden. In diesem Fall müssen die lesenden Aktivitäten entweder durch über das Ereignis $evObjectVersionPreReleased$ bei vorläufigen Daten oder mittels Ereignis $evFinalObjectVersion$ bei endgültigen Objektversionen informiert werden.

3. Für die endgültige Objektversion existiert keine andere Version, die als Ersatz für die lesenden Aktivitäten angeboten werden kann. Die betroffenen Aktivitäten müssen dann über das Ereignis $evFalse$ beziehungsweise $evPreFalse$ benachrichtigt werden. Fehlt ihnen damit ein notwendiges Eingabedatum, so wird bei ihnen die Undo-Konsistenzsicherung angestoßen.

Weitere Details kann der interessierte Leser den UML-Aktivitätsdiagrammen im Anhang B.2 der Arbeit entnehmen. Für die folgenden Überlegungen genügt der in Abbildung 13.15 gezeigte Überblick. In dieser Abbildung ist der endliche Automat aus Abbildung 13.3 um die fehlenden Übergangsbedingungen ergänzt. Zum besseren Verständnis wurde dabei bei jedem einen Zustandsübergang bewirkenden Ereignis auch die dieses Ereignis auslösende Benutzerinteraktion mit angegeben.

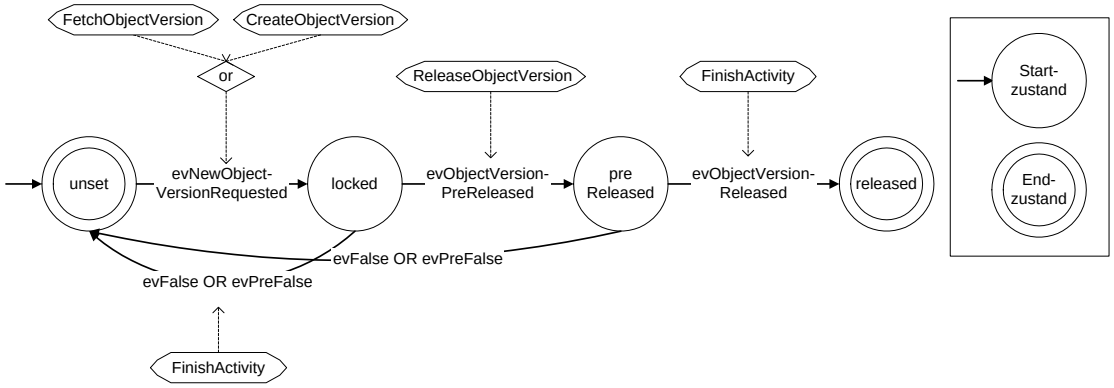



**Abbildung 13.15:** Spezifikation der Übergangsbedingungen von Objektversionszuständen für den endlichen Automaten aus Abbildung 13.3

## 13.7 Erlaubte Benutzerinteraktionen und resultierende Ereignisse

Es sind damit alle relevanten **WEP**-Ereignisse bekannt. Die Abbildung 13.16 greift das Schema aller erlaubten Benutzerinteraktionen aus Abbildung 13.1 auf und zeigt als Zusammenfassung alle Ereignisse, die von einer Interaktion ausgelöst werden. Hierbei sind nicht nur die direkt durch die Benutzerinteraktion ausgelösten Ereignisse dargestellt, sondern auch die Ereignisse, die aus dem Schalten von Kontrollflussknoten und Objektversionen indirekt resultieren.

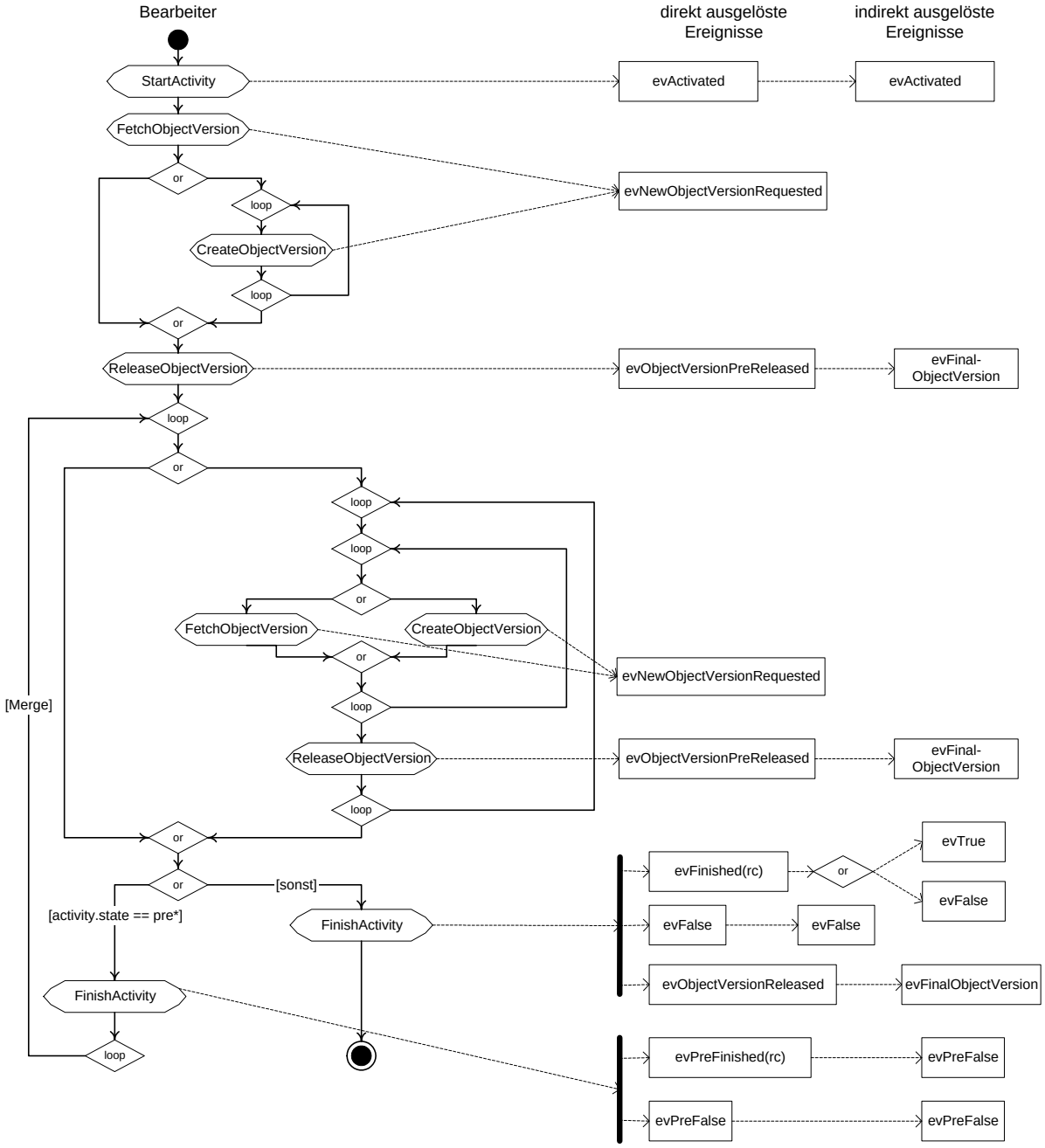



**Abbildung 13.16:** Erlaubte **WEP**-Benutzerinteraktionsfolgen und resultierende Ereignisse

## 13.8 Formale Beschreibung der Aktivitätenzustände und -übergänge

Nun sind alle Voraussetzungen geschaffen, um auch für Aktivitäten die noch nicht spezifizierten Zustandsübergangsbedingungen aus Abbildung 13.2 in Form logischer Verknüpfungen von Ereignissen definieren zu können.

Mit Hilfe der Abbildung 13.16 und der in den vorangegangenen Abschnitten dargelegten Semantik der Ereignisse sowie dem Schaltverhalten der einzelnen **WEP**-Workflow-Graphknoten sind die meisten Übergangsbedingungen für die Aktivitätenzustände auch ohne weitere Erklärung verständlich (siehe Abbildung 13.17). Es werden nun deshalb nur einzelne komplexere Bedingungen genauer erläutert:

### 13.8.1 Übergang zu Zustand $preOffered$

Eine Aktivität wird als abhängige Aktivität angeboten (Zustand $preOffered$), wenn gilt:

*Entweder* ist eine der Vorgängeraktivitäten noch nicht endgültig beendet. Die Aktivität hat dann nur eines der Ereignisse $evActivated$ oder $evPreFinished(rc)$ über ihre Kontrollflusskante erhalten. Ihre notwendigen Eingabedaten können dann in endgültiger oder vorläufiger Form vorliegen, was ihr über die Ereignisse $evFinalObjectVersion$ beziehungsweise $evObjectVersionReleased$ oder $evObjectVersionPreReleased$ signalisiert wird.

*Oder* die Vorgängeraktivitäten sind zwar schon endgültig beendet (Ereignis $evTrue$ oder $evFinished(rc)$). Es gibt aber mindestens einen notwendigen Eingabeparameter, der mit einer noch nicht endgültigen Objektversion verbunden ist.

### 13.8.2 Übergang zu Zustand $preConsRecov$

Die Zustandsübergänge zum Bearbeitungszustand $preConsRecov$, der eine *Konsistenzsicherungsphase* (siehe Kapitel 12) markiert, benötigen ebenfalls eine Erklärung. Eine Aktivität betritt eine Konsistenzsicherungsphase genau dann, wenn:

- die weitere Bearbeitung der Aktivität nicht mehr notwendig ist. Diese Fälle treten bei Verzweigungen auf, wenn feststeht, dass der Zweig, in dem sich die Aktivität befindet, nicht mehr benötigt wird. Es wird dann das Ereignis $evFalse$ oder $evPreFalse$ vom Verzweigungsknoten ► an die erste Aktivität des nicht mehr benötigten Zweigs übermittelt. Bei jeder Aktivität, die sich in den Zuständen $preActivated$ oder $preFinished$ befindet, wird dadurch eine Undo-Konsistenzsicherung angestoßen, welche die von ihr verursachten Änderungen rückgängig macht. Im Rahmen der Konsistenzsicherung werden über ausgehende Kontrollfluss- und Datenflusskanten entsprechend den eingetroffenen Ereignissen entweder $evFalse$- oder $evPreFalse$-Ereignisse weitergeleitet.

  Aufgrund möglicher Datenabhängigkeiten zwischen parallelen Zweigen können dabei auch parallele Zweige betroffen sein: In dem in Abbildung 13.13 gezeigten Beispiel wird der bedingte Zweig mit der Aktivität $A4$ zurückgesetzt, da der alternative Zweig ausgewählt wurde. Die in dieser Abbildung gezeigte ungünstige, aber erlaubte Bearbeitungskonstellation macht deutlich, dass die Übergangsbedingung zum Bearbeitungszustand $preConsRecov$ um den Teil

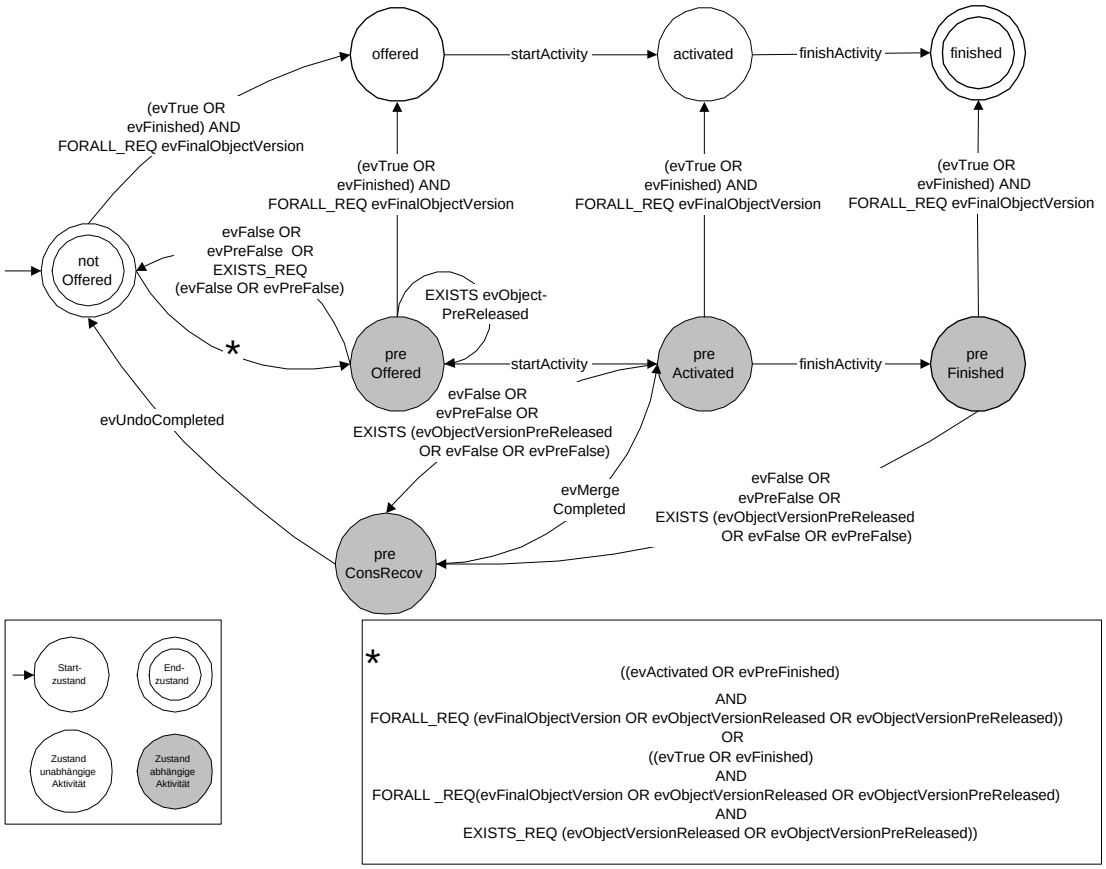



**Abbildung 13.17:** Spezifikation der Übergangsbedingungen von Aktivitätenzuständen für den endlichen Automaten aus Abbildung 13.2. Zur besseren Übersicht wurde bei den Ereignissen $evPreFinished(rc)$ und $evFinished(rc)$ auf die Returncodebezeichnung (*rc*) verzichtet und bei den Übergangsbedingungen die verkürzten Quantifizierungen $FORALL[\_REQ]$ für $\forall\,[required]\ ip \in APINPUTS(A)$ und $EXISTS[\_REQ]$ für $\exists\,[required]\ ip \in APINPUTS(A)$ bei den Datenflussereignissen verwendet.

$\exists\, ip \in APINPUTS(A) : (evFalse\ OR\ evPreFalse)$ erweitert werden muss. Die Konsistenzsicherung kann in allen geschilderten Fällen automatisch erfolgen.

- neue Daten eintreffen (Ereignis $evObjectVersionUpdated$). Je nach Konsistenzsicherungsstrategie erfolgt hier die Konsistenzwiederherstellung automatisch ($acpUndoRedo$, neuer Bearbeitungszustand $notOffered$) oder interaktiv ($acpMerge$, neuer Bearbeitungszustand $preActivated$).

Wiederum sind im Anhang B.2 die UML-Aktivitätsdiagramme zu finden, die im Detail die Ereignisverarbeitung abhängiger Aktivitäten beschreiben.

Damit hat man das eingangs des Kapitels formulierte Ziel, für jeden Aktivitäten- und Objektversionszustand, eine formale Übergangsbedingung festzulegen, erreicht und somit die Grundlagen für den Nachweis dynamischer Eigenschaften von **WEP**-Workflows in Kapitel 14 geschaffen.

# Kapitel 14

# Sicherstellung dynamischer Eigenschaften von WEP-Workflow-Instanzen

Ein Unternehmen wird seine wertschöpfenden Prozesse nur dann einem Workflow-Management-System „anvertrauen“, wenn es garantieren kann, dass es diese Prozesse auch korrekt und blockierungsfrei steuert und diese wohldefiniert terminiert.

Ein Blockieren eines **WEP**-Workflows in Form eines *Deadlocks* wird bereits durch Modellierungsregeln ausgeschlossen. So stellt die strikte Blockstruktur (Abschnitt 7.3) bei dem in Abschnitt 13.5 spezifizierten Schaltverhalten der Kontrollflussknoten und fairen Bearbeitern sicher, dass eine Aktivität immer zur Bearbeitung aus Kontrollflusssicht angeboten werden kann. Die Datenflussanalyse (Kapitel 9) verhindert ein Blockieren aufgrund fehlender Eingabedaten.

Natürlich ist bei Workflow-Modellen mit Schleifen nie ausgeschlossen, dass es zum Blockieren durch Endlosschleifen kommen kann (*Lifelocks*). Dies kann im **WEP**-Modell jedoch durch Spezifikationen geeigneter Zeitrestriktionen erkannt und geeignet eskaliert werden (vergleiche [Kos00]).

Es kann damit im **WEP**-Workflow-Management-System sichergestellt werden, dass ein **WEP**-Workflow „irgendwie“ terminiert. Nicht sicher ist, dass ein **WEP**-Workflow trotz vorzeitiger Datenweitergabe korrekt endet, da möglicherweise die Bearbeitung von Aktivitäten auf Basis noch vorläufiger Daten zu anderen Ergebnissen führen könnte wie eine rein sequentielle Abarbeitung der Aktivitäten ohne jede zeitliche Überlappung. Ziel des restlichen Kapitels ist es nun nachzuweisen, dass dies bei dem spezifizierten Schaltverhalten für **WEP**-Workflows nicht auftreten kann.

Um diesen Nachweis führen zu können, muss zuerst definiert werden, was unter einer korrekten Terminierung von **WEP**-Workflow-Instanzen verstanden wird. Es wird dazu in Anlehnung an die Serialisierbarkeit von Datenbanktransaktionen [Elm92, GR93] der Begriff der *SE-Serialisierbarkeit* als Kriterium für eine korrekte Terminierung von **WEP**-Workflow-Instanzen eingeführt (Abschnitt 14.1). Anschließend wird mit Hilfe des formalen Ausführungsmodells aus Kapitel 13 für beliebige korrekt modellierte **WEP**-Workflows nachgewiesen, dass sie entsprechend der SE-Serialisierbarkeit korrekt terminieren (Abschnitt 14.2).

## 14.1 SE-serialisierbare Terminierung

*Satz 3: SE-serialisierbare Terminierung*

*Die Durchführung einer **WEP**-Workflow-Instanz (mit vorzeitiger Datenweitergabe) führt zum gleichen Ergebnis wie die Durchführung eines analogen Workflows $WF_{ohneSE}$ ohne Möglichkeiten zur vorzeitigen Datenweitergabe: Jede Aktivität des **WEP**-Workflows erzeugt die gleichen Objektversionen als endgültige Ausgabeparameter wie die entsprechende Aktivität des Workflows $WF_{ohneSE}$.*

Die Behauptung in Satz 3 ist natürlich nur wahr, wenn sichergestellt ist, dass die letzte Weitergabe der Ausgabedaten einer Aktivität auf endgültigen Eingabedaten basierte, da dies die Eingaben sind, die auch die entsprechende Aktivität im Workflow $WF_{ohneSE}$ erhalten würde. Das bedeutet, dass eine abhängige Aktivität im **WEP**-Ausführungsmodell zwischen ihrem Angeboten werden (Zustand $preOffered$) und ihrer Beendigung (Zustand $preFinished$) entweder unabhängig (entsprechende nicht-$pre$-Zustände) und damit durch entsprechende Benutzerinteraktionen in den Endzustand $finished$ übergeht oder in ihren Ausgangszustand[1] (Zustand $notOffered$) zurückgesetzt wird.

## 14.2 Beweis der SE-serialisierbaren Terminierung

Aus dem Zustandsübergangsdiagramm aus Abbildung 13.17 erkennt man, dass eine abhängige Aktivität durch die vom Benutzer ausgelösten Interaktionen keinen Endzustand erreichen kann. Ein Übergang in einen Endzustand ist nur durch Eintreffen externer Ereignisse möglich. Bei detaillierter Betrachtung stellt man fest, dass für den letzten Übergang in einen Endzustand die Ereignisse nur von unabhängigen Aktivitäten ausgelöst werden können:

Beim Endzustand $finished$ sind dies die Ereignisse $evTrue$ oder $evFinished(rc)$, die über die Kontrollflusseingangskante signalisiert werden und für jeden notwendigen Eingabeparameter das Ereignis $evFinalObjectVersion$, das über Datenflusseingangskanten gesendet wird (vergleiche auch Abbildung 13.16).

Für den letzten Zustandsübergang in den Endzustand $notOffered$ kommt nur das Ereignis $evFalse$ in Frage, das über Kontroll- oder Datenflusskanten eintreffen kann. Denn nur dann ist sichergestellt, dass die Aktivitäten, die gemäß Kontroll- und Datenfluss Vorgänger der betrachteten Aktivität sind, nicht mehr den in Abbildung 13.17 mit $*$ markierten Zustandsübergang auslösen können.

Eine abhängige Aktivität $A_d$ ist einerseits von Aktivitäten abhängig, die Objektversionen für ihre Eingabeparameter bereitstellen, und anderseits abhängig von ihren Kontrollflussvorgängern. Erstere Aktivitäten sind für jeden Eingabeparameter $ip \in APINPUTS(A_d)$ in der Menge $A_d.IPwriter(ip)$ enthalten (vergleiche Kapitel 11). Aufgrund der Konstruktion von **WEP**-Workflow-Graphen mit expliziten Kontrollflussknoten existiert für jede Aktivität immer genau ein Vorgängerknoten. Dies ist bei einer Sequenz eine zielorientierte Aktivität, in allen anderen Fällen ein Kontrollflussknoten. Der Vorgängerknoten einer Aktivität $A$ wird mit $A.Pred_{cf}$ bezeichnet (siehe Definition in Abschnitt 9.3.1).

Es wird nun mittels Induktion über die Blockstruktur eines **WEP**-Workflows gezeigt, dass für jede abhängige Aktivität $A_d$ ein Bearbeitungszustand des betrachteten **WEP**-Workflows existiert, bei dem $A_d$ nur noch von *unabhängigen* Aktivitäten abhängig ist. Die Mengen $A_d^j.Pred_{cf}$ und $A_d^j.IPwriter^j(ip)$

[1] Dies gilt ebenfalls für alle von der zurückgesetzten Aktivität erzeugten Objektversionen.

($\forall\ ip \in APINPUTS(A_d)$)[2] enthalten in diesem Fall nur unabhängige Aktivitäten. Ist dieser Workflow-Bearbeitungszustand erreicht, so ist sichergestellt, dass die abhängige Aktivität $A_d$ die notwendigen Ereignisse für einen Übergang in einen Endzustand erhält, $A_d$ also unabhängig oder zurückgesetzt wird.

Ausgangspunkt für den Induktionsbeweis ist die Startaktivität $A_s$ eines **WEP**-Workflows. Aufgrund des blockorientierten Aufbaus besteht ein **WEP**-Workflow auf oberster Ebene (Schachtelungstiefe 0) immer aus einer der vier Konstellationen: Sequenz, Schleife, bedingte oder parallele Verzweigung. Ein Aktivitätenknoten repräsentiert damit eine elementare Aktivität oder eine Blockaktivität, die alle Aktivitätenknoten eine Schachtelungstiefe tiefer umfasst. Eine Blockaktivität besitzt als Eingabeparameter alle die Eingabeparameter der von ihr geschachtelten Aktivitäten, die von außerhalb des Blocks mit Objektversionen versorgt werden.

Für den Induktionsstart wird ein **WEP**-Workflow in der Schachtelungstiefe 0 betrachtet und für jede der vier Konstellationen nachgewiesen, dass es einen Bearbeitungszustand der Konstellation gibt, bei dem jede abhängige Aktivität nur von unabhängigen Aktivitäten abhängig ist.

Der Induktionsschritt erfolgt entlang der Schachtelungstiefe. Es wird für jede Blockaktivität der Schachtelungstiefe $k$ gezeigt, dass für alle darin enthaltenen Aktivitäten (Schachtelungstiefe $k+1$) ebenfalls ein Bearbeitungszustand existiert, bei dem sie nur noch von unabhängigen Aktivitäten abhängig sind. Damit ist auch für die Schachtelungstiefe $k+1$ nachgewiesen, dass die Aktivitäten einen Endzustand erreichen können und der Induktionsschritt ist vollbracht.

### 14.2.1 Induktionsstart (Schachtelungstiefe 0)

#### 14.2.1.1 Sequenz

Die Sequenz hat in ihrer allgemeinen Form die in Abbildung 14.1 gezeigte Struktur:

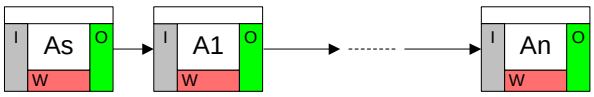



**Abbildung 14.1:** Unabhängigkeitsnachweis Sequenzkette

Der Nachweis, dass es für jede abhängige Aktivität einer Sequenz einen Bearbeitungszustand der Sequenz gibt, bei dem diese Aktivität nur noch von unabhängigen Aktivitäten abhängt, erfolgt per Induktion entlang der Sequenz.

**Induktionsstart:**

Sei $A_1$ abhängige Aktivität. Sie besitzt also einen $pre$-Zustand. Dann gilt für einen korrekt modellierten Workflow:

[2] Der obere Index $j$ gibt für die folgenden Betrachtungen die aktuelle Inkarnation einer Aktivität an. Folgedessen bezeichnet $A_d^j.IPwriter^j(ip)$ die Menge der schreibenden Aktivitäten für den $j$.Aufruf von $A_d^j$. Eine neue Inkarnation wird durch Eintreffen des Ereignisses $evActivated$ gestartet.

$$A_1.IPwriter(ip_i) \subseteq \{A_s\} \forall\, ip_i \in APINPUTS(A_1) \quad (14.1)$$
$$A_1.Pred_{cf} = \{A_s\} \quad (14.2)$$

Da $A_s$ unabhängig ist, folgt aus

*(14.1):* $A_s$ erzeugt die Ereignisfolge $evObjectVersionReleased \rightarrow evFinalObjectVersion$ für jeden Eingabeparameter von $A_1$.

*(14.2):* $A_s$ erzeugt das Ereignis $evFinished(rc)$.

Damit wird $A_1$ von einem beliebigen abhängigen Bearbeitungszustand in einen unabhängigen übergehen (vergleiche Abbildung 13.17) und – einen fairen Bearbeiter vorausgesetzt – den Endzustand $finished$ erreichen.

**Induktionsschritt:**

Sei ein Bearbeitungszustand der Sequenzkette erreicht, bei dem $A_s$, $A_1$, …, $A_j$ unabhängige Aktivitäten darstellen. Dann gilt für einen korrekt modellierten **WEP**-Workflow:

$$A_{j+1}.IPwriter(ip_i) \subseteq \{A_s, A_1, \ldots, A_j\} \forall\, ip_i \in APINPUTS(A_{j+1}) \quad (14.3)$$
$$A_{j+1}.Pred_{cf} = \{A_j\} \quad (14.4)$$

Da die Aktivitäten $A_s$, $A_1$, …, $A_j$ unabhängig sind, folgt aus:

*(14.3):* Es gibt für jeden Eingabeparameter $ip_i \in APINPUTS(A_{j+1})$ eine Aktivität $A_k \in A_s$, $A_1$, …, $A_j$, die die Ereignisfolge $evObjectVersionReleased \rightarrow evFinalObjectVersion$ erzeugt und für die gilt: Es gibt kein $A_l \in A_k.Succ^*_{cf}$, die ebenfalls eine Datenflussabhängigkeit zu $ip_i$ von $A_{j+1}$ besitzt.

*(14.4):* $A_j$ erzeugt Ereignis $evFinished(rc)$.

Damit erhält $A_{j+1}$ alle notwendigen Ereignisse, um von einem beliebigen abhängigen Bearbeitungszustand in einen unabhängigen überzugehen (vergleiche Abbildung 13.17). Sie wird durch entsprechende Benutzerinteraktionen den Endzustand $finished$ erreichen.

■

#### 14.2.1.2 Bedingte Verzweigung

Ausgangspunkt des Beweises ist die in Abbildung 14.2 gezeigte Darstellung einer beliebigen bedingten Verzweigung.

**Betrachtung von $A_1$, …, $A_n$:**

Es sei ein Bearbeitungszustand gegeben, bei dem $A_s$ unabhängige Aktivität ist und sich $A_1$, …, $A_n$ in einem beliebigen $pre$-Zustand befinden. Es gilt für jede Aktivität $A_j \in A_1, \ldots, A_n$:

$$A_j.IPwriter(ip_i) \subseteq \{A_s\} \forall\, ip_i \in APINPUTS(A_j), j = 1, \ldots n \quad (14.5)$$
$$A_j.Pred_{cf} = \{{}_{A_s}\blacktriangleright^{\prec}\} \quad (14.6)$$

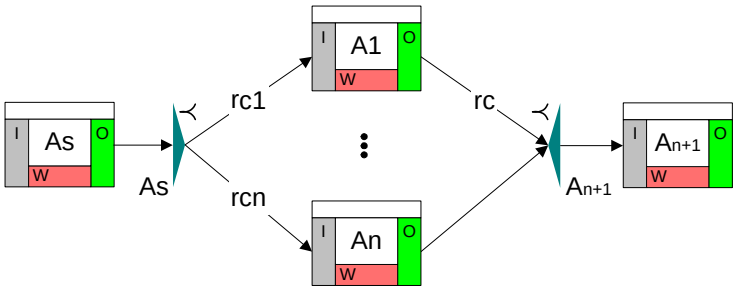



**Abbildung 14.2:** Unabhängigkeitsnachweis bedingte Verzweigung

O. B. d. A. sei der von $A_s$ gewählte Returncode $rc_1$. Damit und aus der Unabhängigkeit von $A_s$ folgt:

*(14.5):* $A_s$ erzeugt die Ereignisfolge $evObjectVersionReleased \rightarrow evFinalObjectVersion$ für alle $ip_i \in APINPUTS(A_1)$.

*(14.6):* $A_s$ erzeugt Ereignis $evFinished(rc_1)$, woraus der Kontrollflussknoten $_{A_s}\blacktriangleright^{\prec}$ über die entsprechenden Kontrollflusskanten das Ereignis

- $evTrue$ an die Aktivität $A_1$ und
- $evFalse$ an alle anderen Aktivitäten $A_2, \ldots, A_n$

sendet (vergleiche Schaltlogik des Kontrollflussknotens im Abschnitt 13.5.2).

Mittels Abbildung 13.17 erkennt man, dass:

1. $A_1$ aus einem beliebigen $pre$-Zustand in einen unabhängigen Bearbeitungszustand übergehen und schließlich den Endzustand $finished$ erreichen wird.
2. $A_2, \ldots, A_n$ per Undo-Konsistenzsicherung in den Endzustand $notOffered$ zurückgesetzt werden.

**Betrachtung von $A_{n+1}$:**

O. B. d. A. wird nun davon ausgegangen, dass ein Bearbeitungszustand des Workflows erreicht wurde, bei dem die Aktivitäten $A_s$ und $A_1$ unabhängig sind ( $A_s$ Bearbeitungszustand $finished$), die anderen Aktivitäten $A_2, \ldots, A_n$ zurückgesetzt wurden und $A_{n+1}$ abhängig ist. (14.7)

Bei korrekt modellierten Workflow gilt außerdem:

$$A_{n+1}.IPwriter(ip_i) \subseteq \{A_s, A_1, \ldots, A_n\} \quad \forall\, ip_i \in APINPUTS(A_{n+1}) \tag{14.8}$$

$$A_{n+1}.Pred_{cf} = \{{}^{\succ}\blacktriangleleft_{A_{n+1}}\} \tag{14.9}$$

Aus (14.7) folgt: $A_1$ sendet das Ereignis $evFinished(rc)$ und $A_2, \ldots, A_n$ jeweils das Ereignis $evFalse$ an den Kontrollflussknoten ${}^{\succ}\blacktriangleleft_{A_{n+1}}$. Knoten ${}^{\succ}\blacktriangleleft_{A_{n+1}}$ ist nach (14.9) Kontrollflussvorgänger

von $A_{n+1}$. Er leitet unabhängig von der Reihenfolge des Eintreffens der Ereignisse immer das Ereignis $evTrue$ weiter. [3]

Betrachtet man 14.8 und die Modellierungsregeln aus Kapitel 9 zur Sicherstellung der Datenversorgung, so gilt für jeden beliebigen Eingabeparameter $ip_i$ von $A_{n+1}$ und unabhängig vom gewählten bedingten Zweig, dass immer mindestens eine unabhängige Aktivität Objektversionen für $ip_i$ bereitstellt. Aus dem Schaltverhalten globaler Objekte (vergleiche Abschnitt 13.6 und insbesondere Abbildung B.8) folgt daraus, dass jeder Eingabeparameter von $A_{n+1}$ das Ereignis $evFinalObjectVersion$ erhält.

Damit erhält $A_{n+1}$ alle notwendigen Ereignisse, um von einem beliebigen abhängigen Bearbeitungszustand in einen unabhängigen überzugehen (vergleiche Abbildung 13.17). Die Voraussetzungen zum Erreichen des Endzustands $finished$ sind damit auch für $A_{n+1}$ erfüllt.

■

#### 14.2.1.3 Parallele Verzweigung

Abbildung 14.3 zeigt die allgemeine Form einer parallelen Verzweigung, auf der der folgende Beweis basiert.

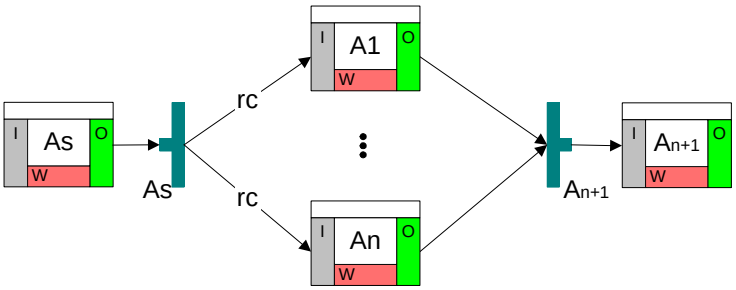



**Abbildung 14.3:** Unabhängigkeitsnachweis parallele Verzweigung

**Betrachtung von $A_1, \ldots, A_n$:**

Es sei ein Bearbeitungszustand des Workflows mit $A_s$ als unabhängige Aktivität gegeben. Dann gilt o. B. d. A. für einen korrekt modellierten Workflow, der insbesondere keine blockierenden Datenflusszyklen enthält:

$$A_j.IPwriter(ip_i) \subseteq \{A_s, A_1, \ldots, A_{j-1}\} \, \forall \, ip_i \in APINPUTS(A_j) \quad (14.10)$$

$$A_j.Pred_{cf} = \{ {}_{A_s}\dashv \} \quad (14.11)$$

Da Aktivität $A_s$ unabhängig ist, erzeugt sie das Ereignis $evFinished(rc)$. Mittels (14.11) und der Schaltlogik von $\dashv$ (siehe Abschnitt 13.5.1) erhält jede der Aktivitäten $A_1, \ldots, A_n$ das Ereignis $evTrue$ über ihre Kontrollflusskante.

[3] Da der Zähler $evActivatedCounter$ im UML-Aktivitätsdiagramm aus Abbildung B.8 größer gleich $n$ (bei $n$ bedingten Zweigen) und nur $n-1$ der Zweige $evFalse$ senden können, wird keines der vor dem $evFinished(rc)$ eintreffenden $evFalse$-Ereignissen weitergeleitet. Dagegen wird das von $A_1$ eintreffende $evFinished(rc)$-Ereignis als $evTrue$ an $A_{n+1}$ gesendet.

Der Nachweis der Unabhängigkeit aller Aktivitäten aus $A_j.IPwriter(ip_i)$ für einen beliebigen Eingabeparameter $ip_i$ von $A_j$ erfolgt über Induktion entlang des Aktivitätenindex $j$:

**Induktionsstart ($j = 1$):**

Aus (14.10) folgt unmittelbar, dass $A_1.IPwriter(ip_i) \subseteq \{A_s\} \ \forall \ ip_i \in APINPUTS(A_1)$. Da $A_s$ unabhängig ist, erzeugt sie die Ereignisfolge $evObjectVersionReleased \rightarrow evFinalObjectVersion$ für jeden Eingabeparameter $ip_i$ von $A_1$.

Damit werden sowohl über die Kontrollflusseingangskanten als auch über die Datenflusseingangskanten die notwendigen Ereignisse erzeugt, damit $A_1$ zu einem unabhängigen Bearbeitungszustand übergehen kann.

**Induktionsschritt ($j \rightarrow j + 1$):**

Es sei ein Bearbeitungszustand des Workflows erreicht, bei dem die Aktivitäten $A_s$, $A_1$, ..., $A_j$ unabhängig sind:

Aus (14.10) folgt:

$$A_{j+1}.IPwriter(ip_i) \quad \subseteq \quad \{A_s, A_1, \ldots, A_j\} \, \forall \, ip_i \in APINPUTS(A_{j+1}) \qquad (14.12)$$

Weil die Aktivitäten $A_s$, $A_1$, ..., $A_j$ unabhängig sind, erzeugen sie für jeden Eingabeparameter von $A_{j+1}$ die Ereignisfolge $evObjectVersionReleased \rightarrow evFinalObjectVersion$. Zusammen mit dem Ereignis $evTrue$ über die Kontrollflusskante sind die Voraussetzungen für den Übergang zu einem unabhängigen Bearbeitungszustand gegeben.

**Betrachtung von $A_{n+1}$:**

Es sei ein Bearbeitungszustand des Workflows gegeben, bei dem $A_s$, $A_1$, ..., $A_j$ unabhängige Aktivitäten sind. Für $A_{n+1}$ gilt außerdem:

$$A_{n+1}.IPwriter(ip_i) \quad \subseteq \quad \{A_s, A_1, \ldots, A_{n+1}\} \, \forall \, ip_i \in APINPUTS(A_{n+1}) \qquad (14.13)$$

$$A_{n+1}.Pred_{cf} \quad = \quad \{ \vdash_{A_{n+1}} \} \qquad (14.14)$$

Durch die Unabhängigkeit von $A_s$, $A_1$, ..., $A_j$ und der in Abschnitt 13.5.1 spezifizierten Schaltlogik von $\vdash$ ist sichergestellt, dass $A_{n+1}$ die benötigten Ereignisse für einen Übergang in einen unabhängigen Bearbeitungszustand erhält.

■

#### 14.2.1.4 Schleife

Eine Schleife in ihrer allgemeinen Form besitzt den in Abbildung 14.4 gezeigten Aufbau:

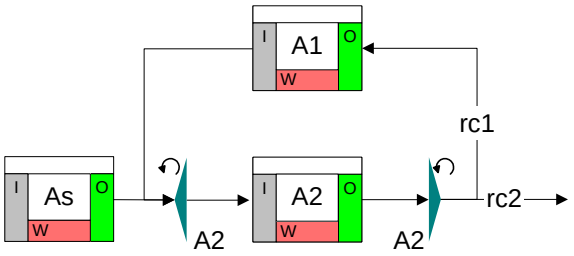



**Abbildung 14.4:** Unabhängigkeitsnachweis Schleife

Beweis erfolgt per Induktion über die Schleifeninkarnationen:

**Induktionsstart: 1. Inkarnation:**

**Betrachtung von $A_2^1$:**

Es sei ein Bearbeitungszustand der Schleife gegeben, bei dem $A_s$ unabhängige und $A_2^1$ abhängige Aktivitäten sind.

Bei der 1. Inkarnation von $A_2^1$ gilt:

$$A_2^1.IPwriter^1(ip_i) \subseteq \{A_s\} \forall\, ip_i \in APINPUTS(A_2^1) \quad (14.15)$$

$$A_2^1.Pred_{cf} = \{{}^{\circlearrowleft}\blacktriangleleft_{A_2}\} \quad (14.16)$$

Aus der Unabhängigkeit von $A_s$ und der Schaltlogik von ${}^{\circlearrowleft}\blacktriangleleft$ (vergleiche Abschnitt 13.5.3) folgt die Unabhängigkeit von $A_2^1$.

**Betrachtung von $A_1^1$:**

Es sei ein Bearbeitungszustand des Workflows gegeben, bei dem $A_s$ und $A_2^1$ unabhängige Aktivitäten sind und $A_1^1$ eine abhängige Aktivität ist. Dann gilt für die 1.Inkarnation von $A_1^1$:

$$A_1^1.IPwriter^1(ip_i) \subseteq \{A_s, A_2^1\} \forall\, ip_i \in APINPUTS(A_1^1) \quad (14.17)$$

$$A_1^1.Pred_{cf} = \{\blacktriangleright^{\circlearrowleft}{}_{A_2}\} \quad (14.18)$$

**Fall 1: Beenden der Schleife:**

$A_2^1$ sendet Ereignis $evFinished(rc_2)$. Entsprechend der Schaltsemantik von ${}^{\circlearrowleft}\blacktriangleleft$ erhält $A_1^1$ das Ereignis $evFalse$ über ihre Kontrollflusseingangskante zugesandt. Nach Abbildung 13.17 wechselt $A_1^1$ in den Endzustand $notOffered$. Weitere Inkarnationen von $A_1$ und $A_2$ gibt es nicht.

**Fall 2: Weiterer Schleifendurchlauf:**

$A_2^1$ sendet Ereignis $evFinished(rc_1)$. Damit erhält $A_1^1$ das Ereignis $evTrue$ über ihre Kontrollflusseingangskante. Aus (14.15) und den Voraussetzung der Unabhängigkeit von $A_s$ und $A_2^1$ folgt die Unabhängigkeit von $A_1^1$.

Für die weitere Betrachtungen ist nur noch der Fall 2 relevant, weil nur hier weitere Schleifendurchläufe auftreten.

**Induktionsschritt $j \rightarrow j+1$:**

**Betrachtung von $A_2^{j+1}$:**

Es sei ein Bearbeitungszustand des Workflows gegeben, bei dem die Aktivitäten $A_s$ und $A_1^j$ unabhängig und $A_2^{j+1}$ abhängig ist. Dann gilt:

$$A_2^{j+1}.IPwriter^{j+1}(ip_i) \subseteq \{A_s\} \forall\, ip_i \in APINPUTS(A_2^{j+1}) \quad (14.19)$$

$$A_2^{j+1}.Pred_{cf} = \{{}^{\circlearrowleft}\blacktriangleleft_{A_2}\} \quad (14.20)$$

Aus der Unabhängigkeit von $A_s$ und $A_1^j$ sowie der Schaltlogik von ${}^{\circlearrowleft}\blacktriangleleft$ folgt die Unabhängigkeit von $A_2^{j+1}$ und der Induktionsschritt ist vollzogen.

**Betrachtung von $A_1^{j+1}$:**

Es sei ein Bearbeitungszustand des Workflows gegeben, bei dem $A_s$ und $A_2^{j+1}$ unabhängige Aktivitäten darstellen und Aktivität $A_1^{j+1}$ abhängig ist. Es gilt:

$$A_1^{j+1}.IPwriter^{1}(ip_i) \subseteq \{A_s, A_2^{j+1}\} \forall\, ip_i \in APINPUTS(A_1^{j+1}) \quad (14.21)$$

$$A_1^{j+1}.Pred_{cf} = \{\blacktriangleright^{\circlearrowleft}_{A_2}\} \quad (14.22)$$

Der Beweis erfolgt analog zur 1. Inkarnation von $A1$.

■

### 14.2.2 Induktionsschritt (Schachtelungstiefe $k \rightarrow k+1$)

Es sei ${}^kA_j$ die erste[4] Blockaktivität in der Schachtelungstiefe $k$ für einen gegebenen **WEP**-Workflow. Da alle anderen Aktivitäten ${}^kA_s$, ${}^kA_1$, $\ldots {}^kA_{j-1}$ in der Schachtelungstiefe $k$ keine Blockaktivitäten sind, gibt es nach Induktionsannahme einen Bearbeitungszustand des Workflows, bei dem ${}^kA_j$ nur noch abhängig von den *unabhängigen* Aktivitäten ${}^kA_s$, ${}^kA_1$, $\ldots {}^kA_{j-1}$ ist.

Betrachtet man die Blockaktivität ${}^kA_j$ in der Schachtelungstiefe $k+1$, so besitzt sie eine der im Induktionsstart beschriebenen Strukturen. Es existiert immer eine eindeutige Startaktivität ${}^{k+1}A_{s_j}$.

Durch die blockorientierte Schachtelung und der Definition der Eingabeparameter von Blockaktivitäten gilt für ${}^{k+1}A_{s_j}$:

$$\forall\, ip_{s_{j_i}} \in APINPUTS({}^{k+1}A_{s_j}) \quad \exists \quad ip_{j_x} \in APINPUTS({}^kA_j): \quad (14.23)$$
$${}^{k+1}A_{s_j}.IPwriter(ip_{s_{j_i}}) = {}^kA_j.IPwriter(ip_{j_x})$$

$${}^{k+1}A_{s_j}.Pred_{cf} = {}^kA_j.Pred_{cf} \quad (14.24)$$

[4] Bei bedingten Verzweigungen wird die Blockaktivität mit dem kleinsten Index zuerst betrachtet. Bei parallelen Verzweigungen immer die Blockaktivität, die keine Leseabhängigkeiten von anderen abhängigen Aktivitäten aus zu ihr parallelen Zweigen besitzt. Für den Beweis werden die Aktivitäten entsprechend umnummeriert.

(14.23) besagt, dass für jeden Eingabeparameter von ${}^{k+1}A_{s_j}$ auch ein Eingabeparameter der ${}^{k+1}A_{s_j}$ unmittelbar umschließenden Blockaktivität ${}^{k}A_j$ existiert, für den die gleichen Aktivitäten Objektversionen bereitstellen (gleiche $IPwriter$-Mengen). Da die $IPwriter$-Menge von ${}^{k}A_j$ nach Induktionsannahme nur unabhängige Aktivitäten enthält (Schachtelungstiefe $k$!), enthält die $IPwriter$-Menge von ${}^{k+1}A_{s_j}$ ebenfalls nur unabhängige Aktivitäten. Daraus folgt, dass jeder Eingabeparameter $ip_{s_{j_i}}$ von ${}^{k+1}A_{s_j}$ das Ereignis $evFinalObjectVersion$ über eine Datenflusseingangskante erhält.

Nach (14.24) besitzen ${}^{k}A_j$ und ${}^{k+1}A_{s_j}$ den gleichen Kontrollflussvorgänger. Wird also ${}^{k}A_j$ unabhängig, weil das Ereignis $evFinished(rc)$ beziehungsweise $evTrue$ über ihre Kontrollflusseingangskante eintrifft, so erhält auch ${}^{k+1}A_{s_j}$ dasselbe Ereignis.

Aktivität ${}^{k+1}A_{s_j}$ erhält also alle für ihre Unabhängigkeit notwendigen Ereignisse. Es ergibt sich damit in der Schachtelungstiefe $k+1$ die gleiche Konstellation wie in der Schachtelungstiefe $k$. Der Induktionsschritt ist dadurch abgeschlossen.

■

# Kapitel 15

# Vorzeitige Datenweitergabe als Grundlage für Simultaneous-Engineering

Vorzeitige Datenweitergabe wird durch die Benutzerinteraktion $ReleaseObjectVersion$ ausgelöst, die bereits in Abschnitt 13.4.4 formal spezifiziert wurde. Ausgeklammert wurde dabei, welche Aktivitäten über eine weitergegebene Objektversion benachrichtigt werden sollen. Dies ist Gegenstand dieses Kapitels.

Die vorzeitige Datenweitergabe zur Unterstützung von Simultaneous-Engineering ist eines der Konzepte des **WEP**-Workflow-Management-Systems, deren korrekte „Handhabung“ zur Laufzeit im allgemeinen Fall sehr komplex ist. Aus diesem Grund wird dieser Aspekt bis zum Detaillierungsgrad von Pseudocode bei den Algorithmen betrachtet (Abschnitt 15.2).

Das Kapitel beginnt mit der detaillierten Beschreibung der Anforderungen, um die verschiedenen bei **WEP**-Workflows auftretbaren Fälle bei der vorzeitigen Datenweitergabe darzulegen.

## 15.1 Anforderungen an eine konsistente vorzeitige Datenweitergabe

Das **WEP**-Workflow-Management-System muss sicherstellen, dass alle Aktivitäten benachrichtigt werden, die potenziell mit diesen Daten noch aktiviert werden können oder bereits aktiviert worden sind. Hierbei muss insbesondere berücksichtigt werden:

- Datenabhängigkeiten zwischen parallelen Zweigen. Zwar ist der Datenaustausch zwischen parallelen Zweigen notwendig. Vorzeitige Datenweitergabe darf dabei nicht zu Dateninkonsistenzen oder Lifelocks führen.
- Im Gegensatz dazu ist der vorzeitige Datenaustausch zwischen Aktivitäten aus verschiedenen bedingten Zweigen in allen Workflow-Konstellationen zu verhindern.
- Bei Schleifen muss sichergestellt werden, dass von einer Aktivität A vorzeitig weitergegebene Daten[1] nicht in ihrem Eingangsbereich (input area) erscheinen oder, falls sie dort erscheinen,

[1] Dabei müssen natürlich auch transitive Abhängigkeiten über weitere globale Objekte berücksichtigt werden.

nicht automatisch die Standardkonsistenzsicherung angestoßen wird (vergleiche Diskussion in Kapitel 12).

Auf diese Anforderungen wird in den nächsten Unterabschnitten detaillierter eingegangen.

### 15.1.1 Datenabhängigkeiten zwischen parallelen Zweigen

Das **WEP**-Modell erlaubt auch den Datenfluss zwischen parallelen Zweigen (vergleiche Abschnitt 9.2). Ausgehend vom folgenden Beispiel in Abbildung 15.1 wird die Problematik erläutert: Die Abbildung zeigt einen Workflow mit 4 Aktivitäten, wobei sich $A2$ und $A3$ in verschiedenen Zweigen einer parallelen Verzweigung befinden. Alle Aktvitäten greifen auf dasselbe globale Objekt zu, $A1$, $A2$, $A3$ schreibend, $A2$, $A3$ und $A4$ lesend. In dieser Abbildung ist folgender Bearbeitungszustand dargestellt: $A1$ hat eine Objektversion weitergegeben und wurde beendet. Basierend auf dieser Version hat $A3$ eine eigene Objektversion abgeleitet und als vorläufiges Datum weitergegeben. Da sowohl $A2$ als auch $A4$ von diesem globalen Objekt lesen, werden sie über die neue Objektversion informiert.

Bei einem angenommenen UndoRedo-Konsistenzsicherungsverfahren von $A2$ wird mit der nun aktuellen Objektversion $v2$ als Eingabe neu gestartet. Sie leitet von dieser Version eine neue ab (Version $v3$) und gibt sie weiter (vergleiche Abbildung 15.2).

#### Variante 1: UndoRedo-Konsistenzsicherungsverfahren

Wurde auch bei $A3$ das UndoRedo-Konsistenzsicherungsverfahren spezifiziert, so muss $A3$ zurücksetzt und mit der neu erhaltenen Objektversion $v3$ erneut gestartet werden. Objektversion $v3$ wird allerdings während der Konsistenzsicherung gelöscht und damit wird die Ursache für das Zurücksetzen der Aktivität entzogen.

Dieses Problem kann vermieden werden, wenn parallele Aktivitäten, zwischen denen Datenabhängigkeiten existieren, durch das **WEP**-Workflow-Management-System geeignet dynamisch *serialisiert* werden. In dem gezeigten Beispiel kann dies erreicht werden, wenn $A2$, die erst nach $A3$ ihre Objektversion weitergibt, nach $A3$ serialisiert wird. $A2$ wird dann abhängige Aktivität von $A3$. Damit darf durch die ReleaseObjektVersion-Operation von $A3$ nur noch die ihr nachfolgende Aktivität $A4$ informiert werden.

#### Variante 2: Merge-Konsistenzsicherungsverfahren

Anders stellt sich die Situation dar, wenn bei Aktivität $A3$ das Merge-Konsistenzsicherungsverfahren spezifiziert wurde. Hier ist das Informieren von $A3$ über die neuen von $A2$ bereitgestellten Eingabedaten sinnvoll, da der Bearbeiter von $A3$ damit „Feedback“ über die von ihm bereitgestellten Daten erhalten kann. Zur Erhaltung der Datenkonsistenz muss ihm allerdings signalisiert werden, dass die neuen Eingabedaten auf von ihm vorläufig weitergegebenen Daten basieren. Nur dadurch kann sichergestellt werden, dass weitergegebene Objektversionen nicht auf Versionen basieren, die später als ungültig markiert werden. Beim Beispiel aus den Abbildungen 15.1 und 15.2 kann also $A3$ noch über die von $A2$ bereitgestellte Version $v3$ informiert werden, obwohl damit eine zyklische Datenweitergabe erzielt wurde. Abweichend vom Standard-Merge-Verfahren darf $A3$ nicht wie üblich vom höchsten für sie sichtbaren ReleaseMarker ( 3, A2 ) Objektversionen ableiten, sondern nur von den mit

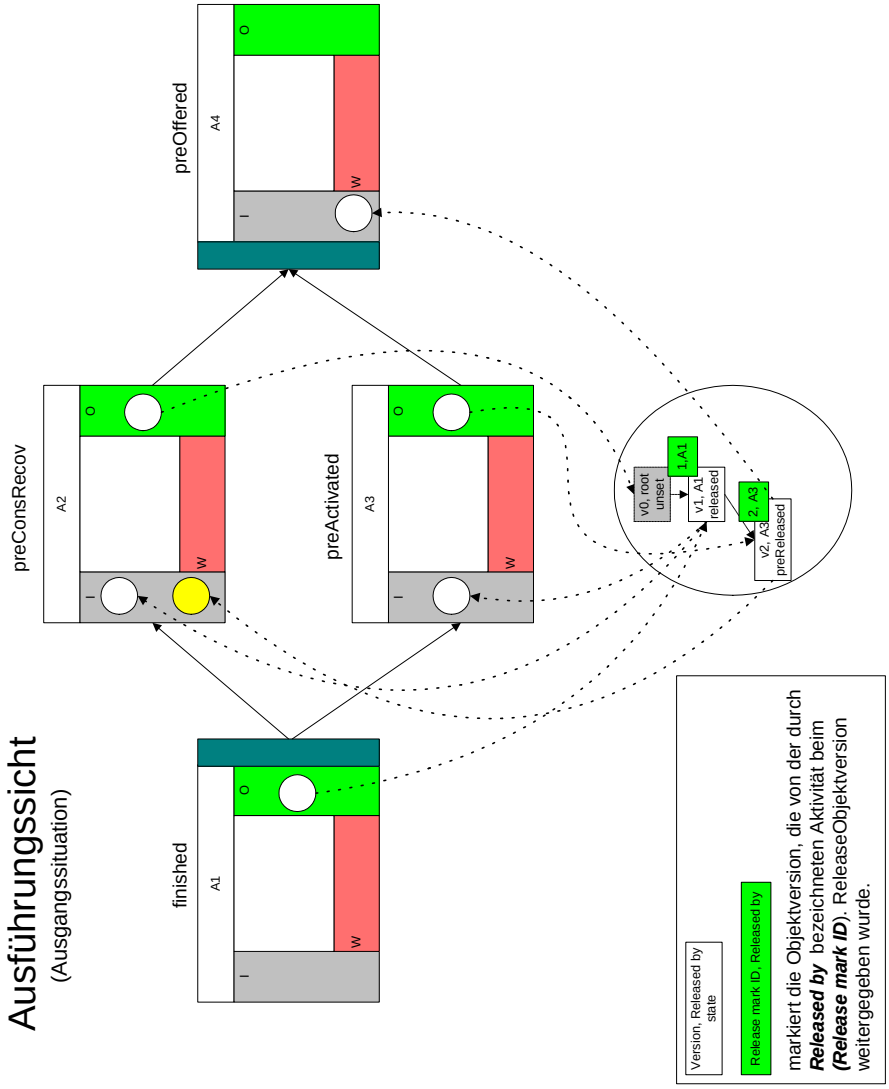



**Abbildung 15.1:** Datenabhängigkeiten zwischen parallelen Zweigen, Ausgangssituation

ReleaseMarkern ( 1, A1 ) und ( 2, A3 ) markierten Versionen. Eine mögliche Folgekonstellation zeigt Abbildung 15.3.

Prozessanalysen haben gezeigt, dass diese Vorgehensweise bei der Variante 2 bei realistischen Anwendungen von vielen Prozessbeteiligten ausdrücklich gewünscht wird. Sie kann jedoch nur beim Merge-Konsistenzsicherungsverfahren vernünftig eingesetzt werden. Ansonsten können Lifelocks entstehen, wie in den Abbildungen 15.1 und 15.2 gezeigt.

Für einen Praxiseinsatz werden also beide Varianten benötigt. Sie unterscheiden sich jedoch nur marginal bei der Bestimmung der zu benachrichtigenden Aktivitäten. In beiden Varianten muss die zyklische Datenweitergabe erkannt werden. Bei Variante 1 wird die neue Objektversion der „letzten“ Aktivität, die zum Zyklus führen würde, nicht mehr bereitgestellt. Bei der Variante 2 wird sie über die neue Objektversion zwar informiert. Es wird aber nicht das Standard-Merge-Verfahren angewandt, weil andere Objektversionsableitungsregeln existieren (vergleiche Abbildung 15.3).

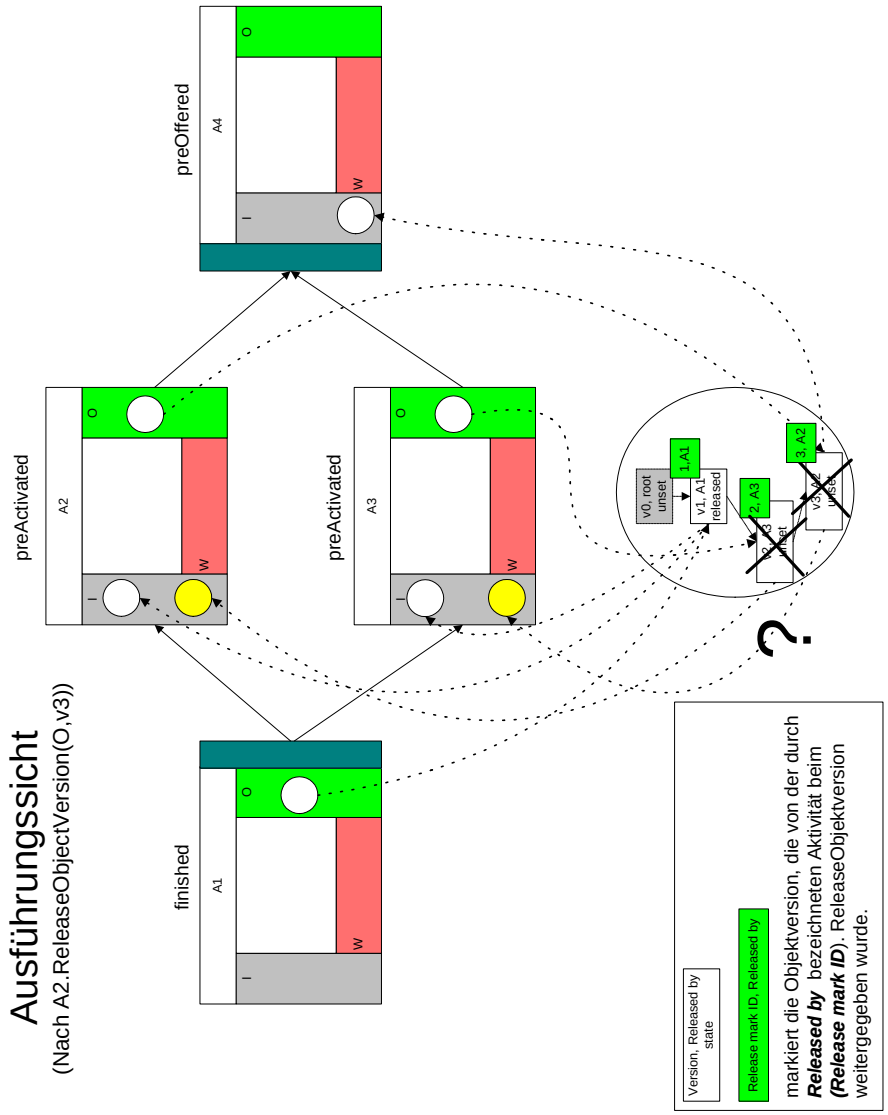



**Abbildung 15.2:** Datenabhängigkeiten zwischen parallelen Zweigen, nach der Benutzerinteraktion $A2.ReleaseObjectVersion(O, v3)$)

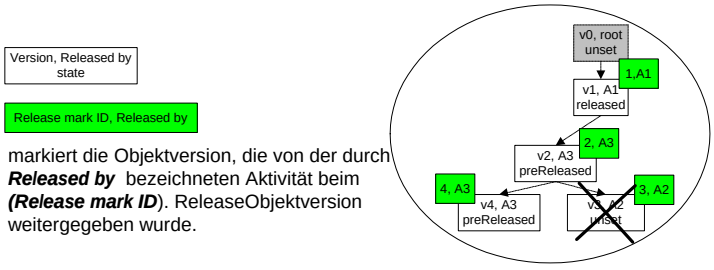



**Abbildung 15.3:** Datenabhängigkeiten zwischen parallelen Zweigen, nach Durchführung des Merge-Konsistenzsicherungsverfahrens

Da es in diesem Kapitel hauptsächlich um die Bestimmung der bei einer vorzeitigen Datenweitergabe betroffenen Aktivitäten geht, wird vorerst aufgrund ihrer geringeren Komplexität der ersten Variante der Vorzug gegeben. Bei einem Wechsel zur Variante 2 sind nur geringe algorithmische Änderungen notwendig, die für das weitere Verständnis belanglos sind.

### 15.1.2 Datenabhängigkeiten zwischen bedingten Zweigen

Im Gegensatz zu einer parallelen Verzweigung ist ein Datenaustausch zwischen verschiedenen Zweigen einer bedingten Verzweigung nicht sinnvoll (vergleiche auch Datenflussanalyse in Kapitel 9). Diese Konstellation tritt ein, wenn – wie in Abbildung 15.4 gezeigt – durch vorzeitige Datenweitergabe sowohl $A2$ als $A3$ parallel arbeiten können. Gibt eine der beiden Aktivitäten ihrerseits eine Objektversion weiter, so dürfen Aktvitäten in anderen bedingten Zweigen nicht benachrichtigt werden, da am Ende immer nur einer der Zweige durchlaufen wird. Die anderen Zweige werden zurückgesetzt. Die Datenweitergabe erfolgt deshalb nur entlang des Kontrollflusses (in Abbildung 15.4 an Aktivität $A4$). Das im nächsten Abschnitt vorgestellte Verfahren muss diese Semantik der vorzeitigen Datenweitergabe bei bedingten Verzweigungen geeignet umsetzen.

### 15.1.3 Datenabhängigkeiten bei Schleifen

Das **WEP**-Workflow-Management-System muss erkennen, wenn – bedingt durch Schleifen – vorzeitige Datenweitergabe wieder bei der noch nicht beendeten Ursprungsaktivität ankommt (vergleiche Abbildung 15.5), um zu verhindern, dass die Konsistenzsicherung bei der Ursprungsaktivität angestoßen wird. Die Konsistenzsicherung bei Schleifen muss deshalb besonders behandelt werden. Es sind folgende Varianten denkbar:

**Variante 1:**

Wird erkannt, dass eine Aktivität Daten erhalten soll, die direkt oder indirekt auf Daten basieren, die diese Aktivität vorläufig weitergereicht hat, so wird die Weitergabe an diese Aktivität verhindert.

**Variante 2:**

Bei dieser Variante wird die Weitergabe zwar zugelassen. Es wird aber – analog zur Variante 2 bei parallelen Zweigen – nicht automatisch eine Konsistenzsicherungsphase angestoßen. Stattdessen wird der Aktivität signalisiert, dass diese auf Daten basieren, die sie direkt oder indirekt weitergegeben hat. Der Bearbeiter der Aktivität entscheidet dann individuell, ob eine Konsistenzsicherung angestoßen wird oder ob er verbesserte Objektversionen weitergibt, welche die neuen Eingabedaten mit berücksichtigen.

Bei Prozessanalysen in Entwicklungsbereichen stellte sich heraus, dass einem Bearbeiter solche „Feedback-Informationen“, wie sie mittels Variante 2 realisierbar sind, auch bei Schleifen sehr wohl von Nutzen sind. Variante 2 ist jedoch nur sinnvoll beim Merge-Konsistenzsicherungsverfahren anwendbar, so dass bei Auswahl der Variante 2 die erste Variante für das UndoRedo-Konsistenzsicherungsverfahren zusätzlich benötigt wird. Die folgenden Algorithmen konzentrieren sich deshalb wiederum auf die Variante 1.

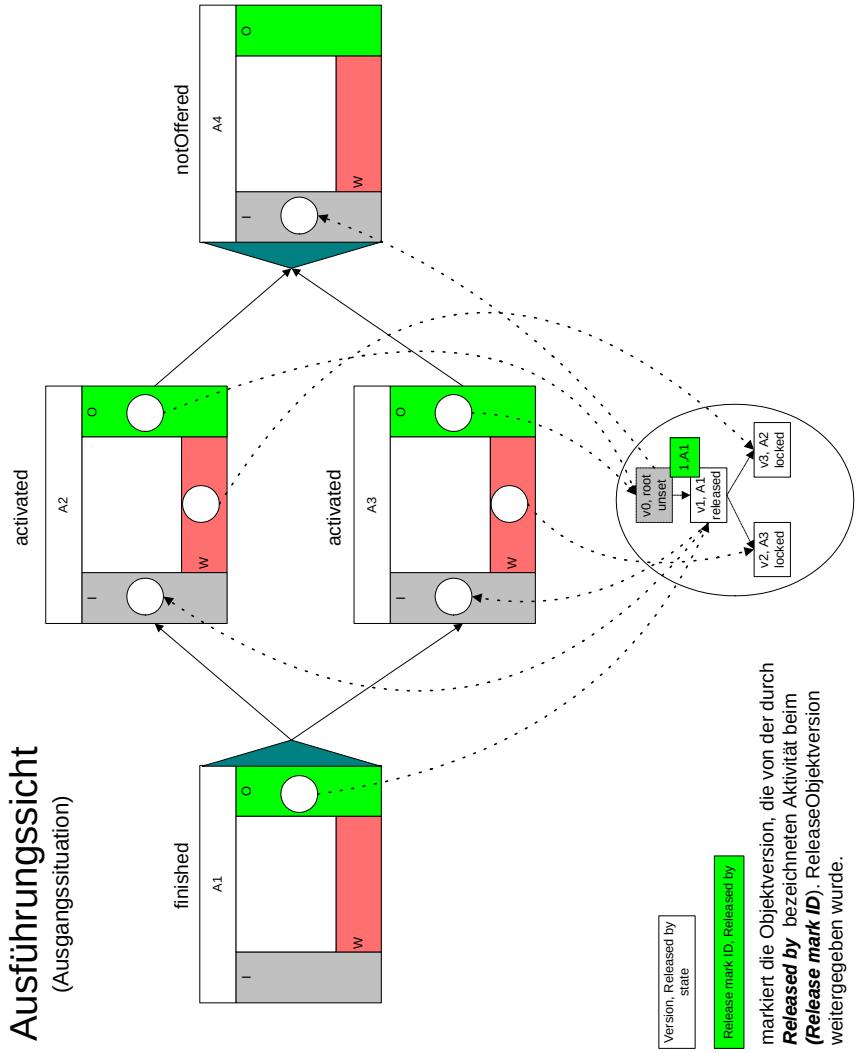



**Abbildung 15.4:** Datenabhängigkeiten zwischen bedingten Zweigen, Ausführungssicht

## 15.2 Realisierung vorzeitiger Datenweitergabe

Dieser Abschnitt beschreibt, wie die geschilderten Anforderungen an eine konsistente vorzeitige Datenweitergabe algorithmisch im **WEP**-Workflow-Management-System umgesetzt werden.

### 15.2.1 Voraussetzungen und grundlegende Definitionen

Eine effiziente vorzeitige Datenweitergabe setzt voraus, dass die Laufzeitalgorithmen auf einer geeigneten internen Darstellung eines **WEP**-Workflows aufsetzen können. Diese Laufzeitdarstellung wird sich – aufgrund der verschiedenen Anforderungen von Modellierungs- und Laufzeitkomponente – von der Darstellung zur Modellierungszeit unterscheiden. Dies ist allerdings unproblematisch, wenn sich aus der Modellierungsdarstellung des **WEP**-Workflows die Laufzeitdarstellung ableiten lässt und diese Ableitung automatisch nach Abschluss der Modellierungsphase durchgeführt werden kann.

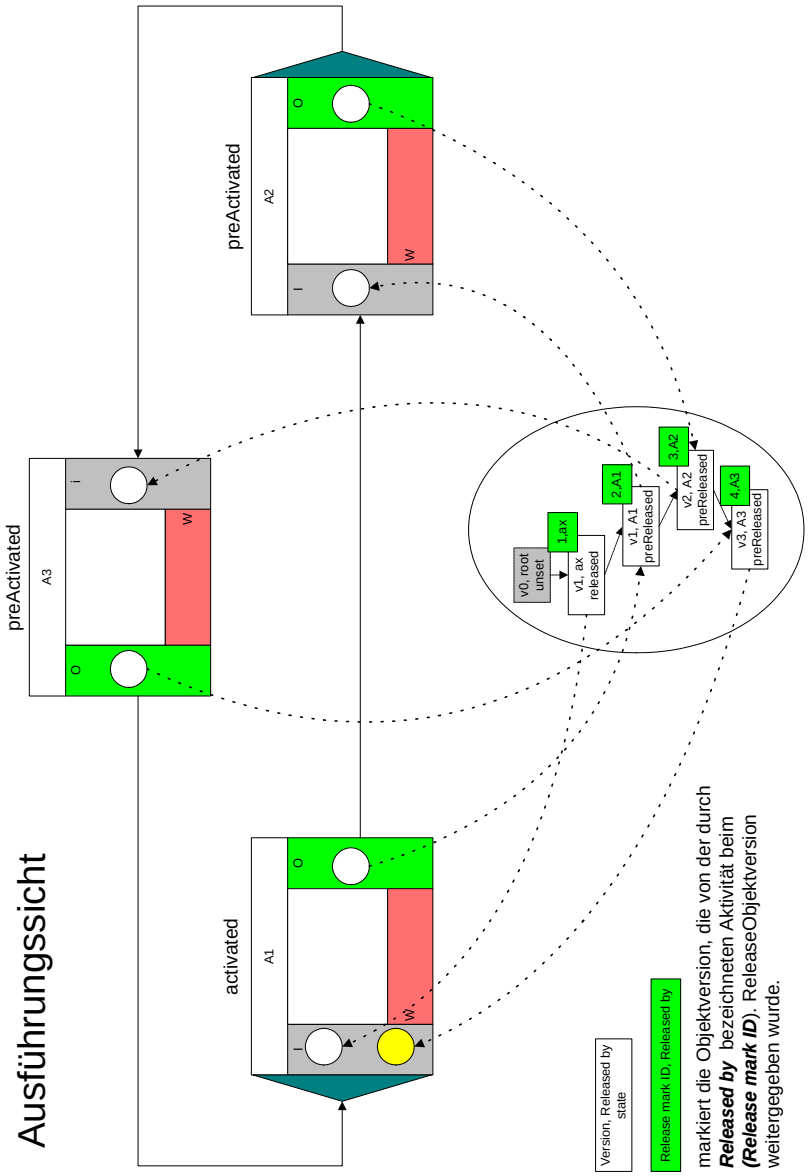



**Abbildung 15.5:** Datenabhängigkeiten bei Schleifen. Weitergabe vorläufiger Daten an die Ursprungsaktivität: Objektversion $v4$ wird an $A1$ weitergegeben; Version $v4$ wurde von Objektversion $v1$ abgeleitet (transitiv), die ursprünglich von $A1$ weitergegeben wurde.

Bei dem als nächstes beschriebenen Workflow-Graph **WEP**-$workflow_{ROV}$ treffen diese Bedingungen zu. Er repräsentiert die Darstellung eines **WEP**-Workflows, wie er zur Laufzeit für eine effiziente Durchführung der vorzeitigen Datenweitergabe benötigt wird. An dieser Stelle wird dabei nur exemplarisch gezeigt, wie sich dieser Graph aus der Modellierungsbeschreibung aufbauen lässt. Detaillierte Algorithmen mit Korrektheitsbeweisen sind im Anhang B.1 zu finden. Abschnitt 15.2.2 beschreibt danach Laufzeitalgorithmen, die diesen Graph voraussetzen.

Ein Workflow-Graph **WEP**-$workflow_{ROV}$ enthält für jede seiner Aktivitäten $A$ und jeden ihrer Ausgabeparameter $op \in APOUTPUTS(A)$ die aufbereiteten Informationen *Parallelitätsgraph* und *potenzielle Aktivierungsmenge*, die sich aus dem modellierten Workflow ableiten lassen:

- Der Parallelitätsgraph $parGraph(A, op, oql)$ enthält als Knoten im Wesentlichen alle Aktivitäten, die auf zu $A$ parallelen Pfaden liegen, und über Datenflusskanten direkt mit diesem Ausgabeparameter verbunden sind. Die Reihenfolge der Aktivitäten ergibt sich aus ihrer Kontrollflussreihenfolge. Befindet sich eine Aktivität innerhalb einer Parallelität, so existiert für jeden ihrer Ausgabeparameter und für jeden der dort spezifizierten Qualitätsstufen $oql$ ein solcher Parallelitätsgraph.
- Eine potenzielle Aktivierungsmenge $potActSet^{df}_{ROV}(A, op, oql)$: Diese Menge enthält alle gemäß Kontrollfluss nachfolgenden Aktivitäten, die von dem globalen Objekt in der Qualitätsstufe $oql$ lesen, in das die Aktivität $A$ über den Ausgabeparameter $op$ schreibt und für die $A$ die letzte schreibende Aktivität ist.

Die Abbildungen 15.6 und 15.7 zeigen Beispiele für Parallelitätsgraphen und für $potActSet^{df}_{ROV}$-Mengen. Es wird dabei davon ausgegangen, dass alle Aktivitäten nur einen Ein- sowie Ausgabeparameter besitzen und alle auf das gleiche globale Objekt zugreifen.

### 15.2.2 Bestimmung der zu benachrichtigenden Aktivitäten zur Laufzeit

Der Workflow-Graph **WEP**-$workflow_{ROV}$ ist der Ausgangspunkt für die Durchführung der eigentlichen $ReleaseObjectVersion$-Aktion zur Laufzeit. Der Algorithmus dieser Aktion wird nun schrittweise hergeleitet und anschließend seine Korrektheit nachgewiesen.

#### 15.2.2.1 Der Aktivitätenabhängigkeitsgraph

Mit den soeben beschriebenen Aufbau des Workflow-Graphen **WEP**-$workflow_{ROV}$ sind alle Voraussetzungen geschaffen, um zur Laufzeit die Menge der Aktivitäten bestimmen zu können, die über eine neue Objektversion benachrichtigt werden müssen. Lässt man die in Kapitel 15 beschriebenen zyklischen Datenabhängigkeiten außer Acht, so hat man mit den Aktivitäten aus der Menge $potActSet^{df}_{ROV}$ und dem Graph $parGraph$ bereits alle Aktivitäten bestimmt, die benachrichtigt werden müssen.

Um zyklische Datenabhängigkeiten zu erkennen beziehungsweise gegebenenfalls auch zu vermeiden, müssen hier jedoch weitere Einschränkungen gemacht werden. Dazu wird zur Laufzeit für jedes Objekt $O$ ein Aktivitätenabhängigkeitsgraph aufgebaut (Erweiterung auf mehrere Objekte siehe Abschnitt 15.2.4), der protokolliert, welche Aktivität an welche andere Aktivitäten zuletzt Daten weitergegeben hat:

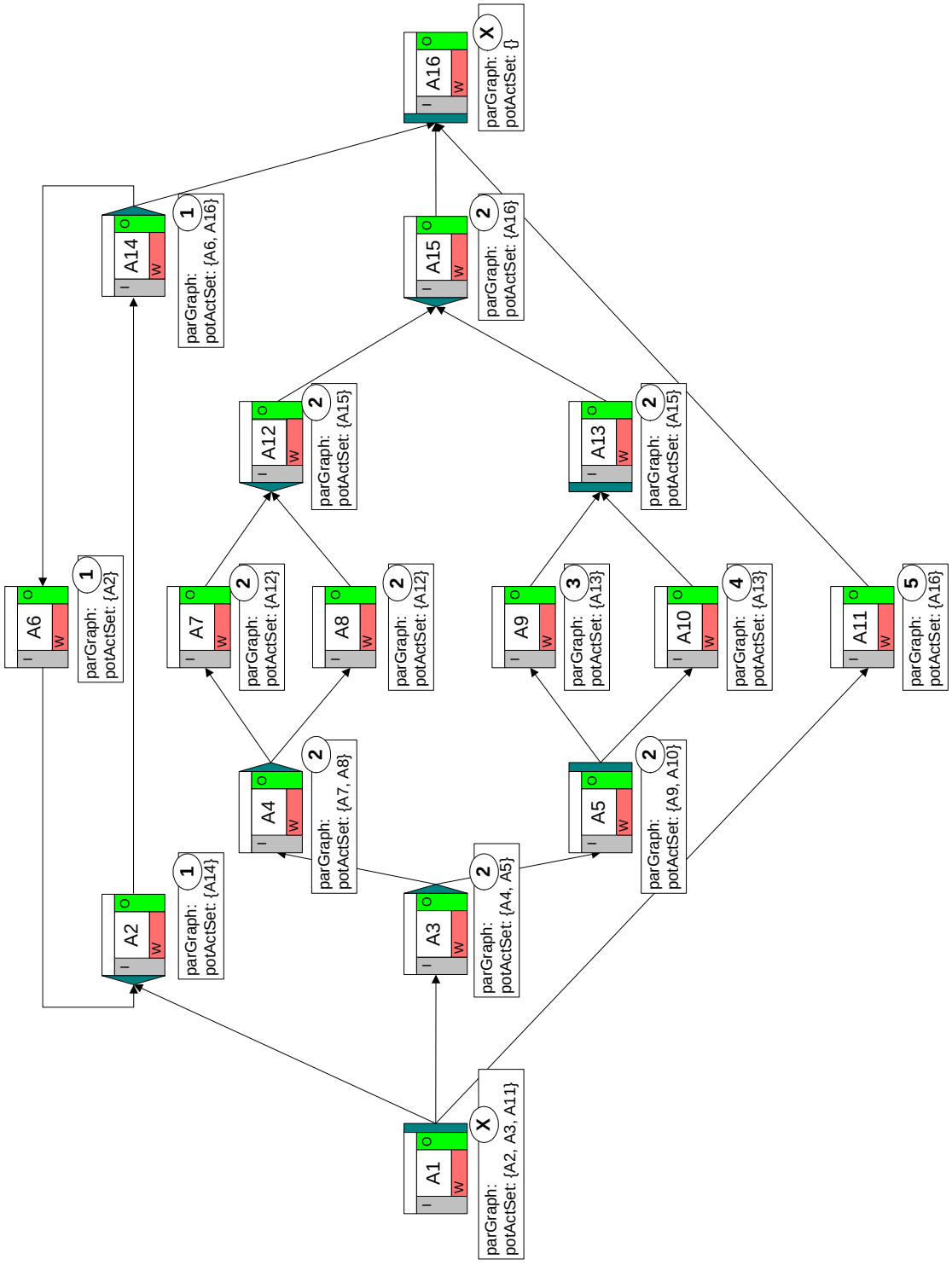



**Abbildung 15.6:** Jeder der drei parallelen Zweige besitzt einen Parallelitätsgraph (siehe entsprechende Nummer in Abbildung 15.7), der die Aktivitäten und Kontrollflusskonstrukte der anderen parallelen Zweige enthält. Für jede Aktivität wird außerdem ihre $potActSet^{df}_{ROV}$-Menge gezeigt, die ihre lesenden Nachfolgeraktivitäten gemäß Kontrollfluss enthält.

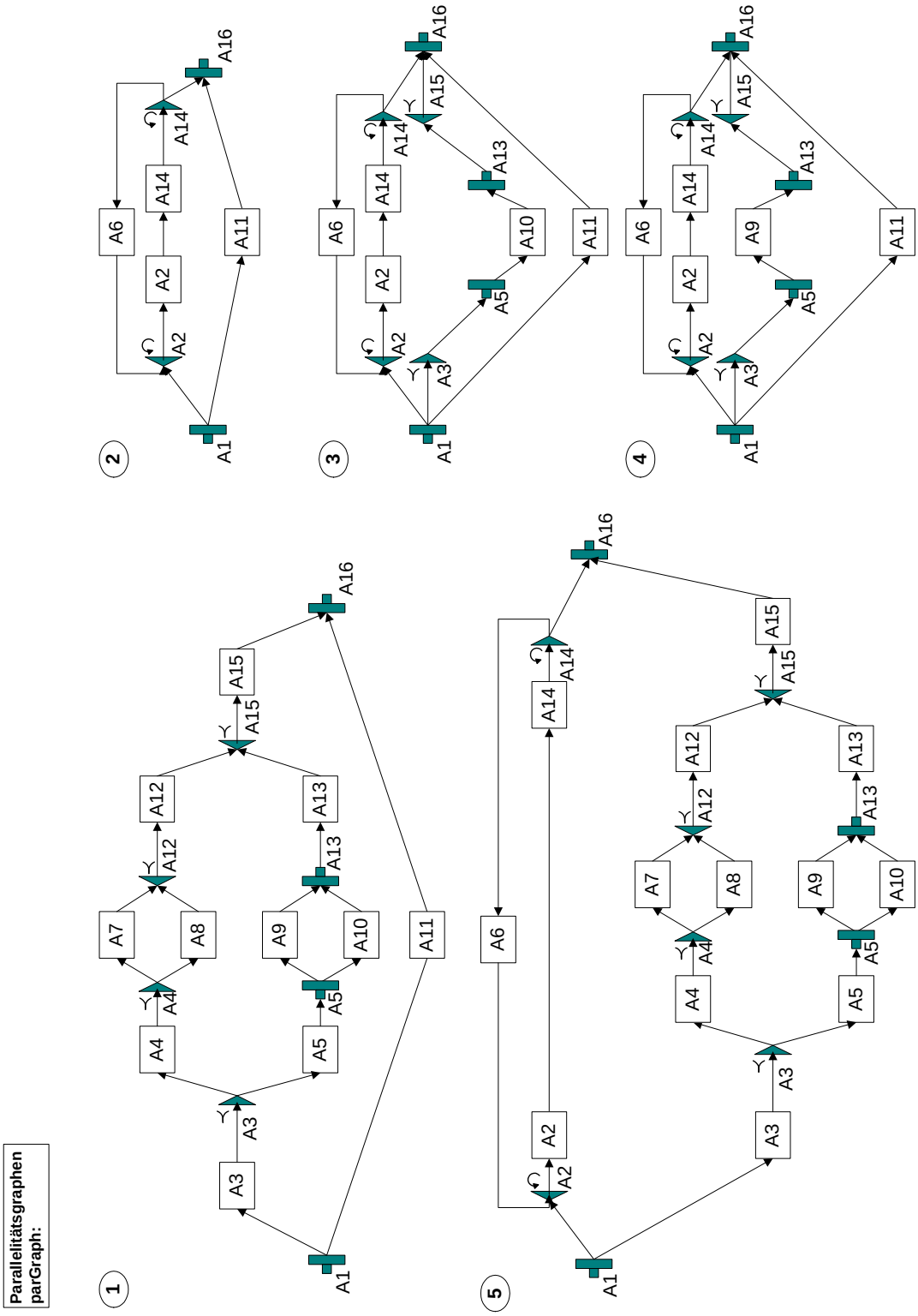



**Abbildung 15.7:** Parallelitätsgraphen zu dem Workflow-Graph aus Abbildung 15.6

Der *Aktivitätenabhängigkeitsgraph* (*activity dependence graph*, $actDepGraph$ ) eines Objekts $O$ $actDepG(O) = (AS(O), ES)$ ist definiert:

*Definition 11:* $actDepG(O) = (AS(O), ES)$

- $AS(O)$ Menge aller Aktivitäten eines Workflows $WF$, die in dieses Objekt schreiben beziehungsweise von diesem Objekt lesen und ein ausgezeichnetes Wurzelobjekt $root$:

$$AS(O) := \{A \in ACTIVITIES(WF) : O \in outputObject(A) \vee O \in inputObject(A)\} \cup root^2$$

- $ES$ Menge von Kanten $E$ zwischen Aktivitäten aus $AS(O)$, für die gilt:

$$E \in ES : \Leftrightarrow Ai, Aj \in AS(O) \wedge Ai \text{ hat Objektversion an } Aj \text{ weitergegeben.}$$

Die Abbildung 15.8 zeigt die Abhängigkeitsgraphen für die in den Abbildungen 15.1 und 15.2 gezeigten Bearbeitungszustände eines **WEP**-Workflows.

Die nun folgenden Algorithmen stellen zur Laufzeit sicher, dass der Aktivitätenabhängigkeitsgraph zyklenfrei bleibt, in dem nur die Aktivitäten aus $parGraph(A, op, oql)$ beziehungsweise $potActSet^{df}_{ROV}(A, op, oql)$ über neue Objektversionen informiert werden, die nicht auf direkter Linie von der Wurzel des Aktivitätenabhängigkeitsgraphen zu der weitergebenden Aktivität $A$ erreicht werden können. Durch die gewählte Kantendefinition im Abhängigkeitsgraph stellt die Zyklenfreiheit im Aktivitätenabhängigkeitsgraph auch eine zyklenfreie Datenweitergabe im Workflow-Graph sicher, weil Aktivitäten, die ihrerseits an $A$ eine Objektversion weitergegeben haben, nicht informiert werden.

Die Bestimmung der zu benachrichtigenden Aktivitäten gliedert sich in drei Hauptschritte: Zuerst wird untersucht, ob die weitergebende Aktivität bereits im Aktivitätenabhängigkeitsgraph enthalten ist. Dies ist immer dann der Fall, wenn die Aktivität bereits zuvor ein $ReleaseObjectVersion$ durchgeführt hat oder schon eine Objektversion von einer anderen Aktivität erhalten hat. Andernfalls wird sie in den Aktivitätenabhängigkeitsgraph unter die Wurzel $root$ gehängt. Dieses Vorab-Einfügen einer noch nicht im Abhängigkeitsgraph vorhandenen Aktivität vereinfacht die anderen Algorithmen, weil weniger Fallunterscheidungen getroffen werden müssen. In beiden Fällen wird die gegenwärtige Position der Aktivität im Abhängigkeitsgraph in der Variablen *aADG* gespeichert, die im zweiten Hauptschritt an den Algorithmus *FindActForNotification* übergeben wird.

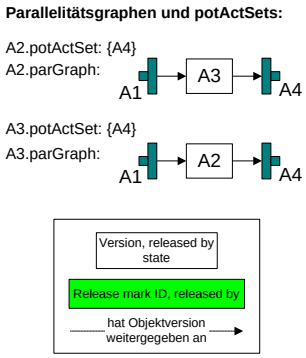
*FindActForNotification* bestimmt alle zu benachrichtigenden Aktivitäten (siehe Abschnitt 15.2.2.2).

Im dritten und letzten Schritt wird der Abhängigkeitsgraph umgebaut, damit er wieder die aktuelle Weitergabesituation widerspiegelt (vergleiche Abschnitt 15.2.2.4).

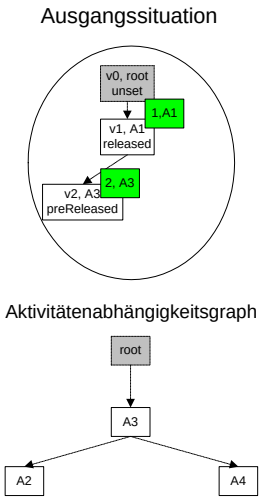
#### 15.2.2.2 Der Algorithmus *FindActForNotification*

Der rekursiv aufgebaute Algorithmus *FindActForNotification* erhält beim Aufruf als Eingabeparameter den Knoten *aADG*, der im aktuell gültigen Aktivitätenabhängigkeitsgraph $actDepGraph(O)$ des Objekts $O$ die auslösende Aktivität $A$ repräsentiert. Zusätzlich wird der Aktivitätenabhängigkeitsgraph selbst, der der Aktivität *aADG* und dem Objekt $O$ zugeordnete Parallelitätsgraph sowie die

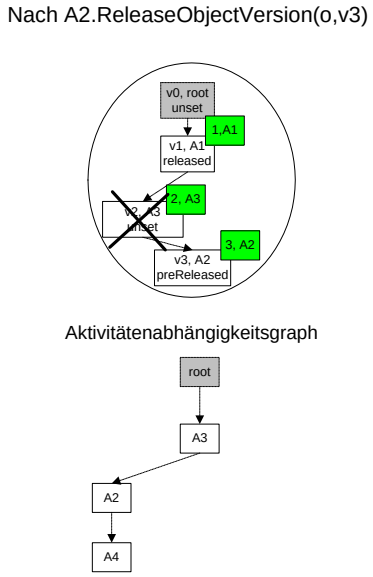
[2] $inputObject$ beziehungsweise $outputObject$ bezeichnet die Menge der globalen Objekte, die mit einem Ein- beziehungsweise Ausgabeparameter von $A$ verbunden sind.




(a) Parallelitätsgraphen und potActSets




(b) Aktivität $A1$ ist beendet. Sie wurde deshalb aus dem Abhängigkeitsgraph entfernt. $A3$ hat als letzte Aktivität eine Objektversion weitergegeben. Da weder $A2$ noch $A4$ im (leeren) Abhängigkeitsgraph als in direkter Linie verwandte Vorfahrin zu $A3$ zu finden sind, kann Aktivität $A3$ beide über die neue Objektversion informieren.




(c) Aktivität $A3$ befindet sich im Abhängigkeitsgraph als in direkter Linie verwandte Vorfahrin zu $A2$. Damit wird nur $A4$ informiert und im Abhängigkeitsgraph umgehängt, weil sie nun die neueste Objektversion von $A3$ erhalten hat.

**Abbildung 15.8:** Abhängigkeitsgraphen für die in den Abbildungen 15.1 und 15.2 gezeigten Bearbeitungszustände des **WEP**-Workflows

weiterzugebende Objektversion $ov$ übergeben. *FindActForNotification* liefert als Ergebnis eine Menge von Aktivitäten zurück, die alle über die neue Objektversion zu benachrichtigen sind. Er berücksichtigt dabei den jeweiligen Release-Status des Workflows bezüglich der weitergebenden Aktivität $A$. In der ersten Phase, repräsentiert durch die interne Methode *FindActForNotificationPG*, werden dabei mittels Parallelitätsgraph alle Aktivitäten bestimmt, die auf zu der betrachteten Aktivität $A$ parallelen Zweigen liegen und über eine Datenflusseingangskante mit dem Objekt $O$ verbunden sind.

- *FindActForNotificationPG* durchläuft damit für eine Aktivität $A$ jeden Pfad des zugeordneten Parallelitätsgraphen und liefert als Ergebnismenge immer die erste Aktivität eines Pfades zurück, die *nicht* ihrerseits bereits an $A$ eine Objektversion für dieses Objekt weitergegeben hat. Letzteres stellt er über die Untersuchung des gegenwärtigen Zustands des Aktivitätenabhängigkeitsgraphen fest. Hat eine Aktivität $Ai$ an $A$ eine Objektversion weitergegeben, so ist $Ai$ im zugeordneten Aktivitätenabhängigkeitsgraph eine in direkter Linie verwandte Vorfahrin von $A$. Die im Algorithmus verwendete Methode *IsDependent* liefert dann ***true***.
- Befindet sich die Aktivität $A$ in einem Zweig einer bedingten Verzweigung, so werden Aktivitäten aus anderen bedingten Zweigen nicht in die Ergebnismenge aufgenommen. Analoges gilt für Schleifen, die im Parallelitätsgraph wie bedingte Verzweigungen behandelt werden können.
- Aktivitäten außerhalb von Parallelitätskonstrukten besitzen keinen Parallelitätsgraph. Der Algorithmus *FindActForNotificationPG* liefert in diesen Fällen die leere Menge als Ergebnis zurück.

In der zweiten Phase des Algorithmus wird die bisher gefundene Aktivitätenmenge – im Folgenden mit $notifSet_{ROV}(A, op, oql)$ bezeichnet – um alle Aktivitäten aus $potActSet^{df}_{ROV}(A, op, oql)$ ergänzt, die nicht ihrerseits bereits an $A$ eine Objektversion weitergegeben haben.

*FindActForNotification (aADG: A, adg: $actDepGraph$, nPG: $ACTIVITIES^{ext}$, ov: O): $\wp(A)$*

```
// interne Methode zur Bestimmung der Aktivitäten aus zu aADG parallelen Zweigen
FindActForNotificationPG (aADG: A, adg: actDepGraph, nPG: ACTIVITIES^ext ): ℘(A)
begin
  NFS := ∅;        // NFS = Notification Set
  if (nPG.isEmpty == false) then
    switch nPG:
      case nPG == ┫  or nPG == ┫^tv  or nPG == ▶⟳ :
        // Untersuche, ob Aktivitäten aus diesen parallelen Zweigen oder Schleifen
        // bereits ReleaseObjectVersion durchgeführt haben.
        for (aPG ∈ nPG.Succ) do                                          ⊛⊛⊛⊛
          NFS := NFS ∪ FindActForNotificationPG(aADG, adg, aPG);          ⊛
        endfor
      case nPG == ▶≺ :
        // Untersuche, ob einer der bedingten Zweige bereits ReleaseObjectVersion durchgeführt hat.
        // Speichere die erste Aktivität dieses Zweiges in hit.
        hit := ∅;
        for ((nPGh ∈ nPG.Succ) and (hit == ∅)) do
          if (nPGh ∈ A) then
            if (adg.IsDependent(aADG, nPGh) == true) then
              // adg.IsDependent(aADG, nPGh) liefert true, wenn aADG im Aktivitätenabhängigkeits-
              // graph ein in direkter Linie verwandter Nachfolger von nPGh ist, also aADG von nPGh
              // eine Objektversion erhalten hat. Die erste Aktivität aus diesem bedingten Zweig hat
              // bereits ReleaseObjectVersion durchgeführt. adg.IsDependent(aADG, nPGh) liefert auch
              //  false, wenn der Aktivitätenabhängigkeitsgraph leer ist bzw. nPGh nicht enthalten ist.
              hit := nPGh;                                               ⊛⊛⊛⊛⊛
            endif;
          else
```

```
            // Schleifenanfang; untersuche die erste Aktivität in der Schleife.
            if ( adg.IsDependent(aADG, nPGh.Succ) == true) then
              hit := nPGh;                                                    ⊛⊛⊛⊛⊛
            endif;
          endif;
        endfor;
        if ( hit == ∅) then
          // Keine der Anfangsaktivitäten aus den bedingten Zweigen hat bereits
          // ReleaseObjectVersion durchgeführt. → Nimm diese als Treffer auf.
          for ( nPGh ∈ nPG.Succ) do                                           ⊛⊛⊛⊛
            // ist Nachfolger von nPG das Symbol für Schleifenbeginn
            if ( nPGh == ⟲ ) then
              if ( nPGh.Succ.isDummyActivity == false ) then
                NFS := NFS ∪ nPGh.Succ;                                       ⊛⊛⊛
              endif;
            else
              if ( nPGh.isDummyActivity == false) then
                NFS := NFS ∪ {nPGh};                                          ⊛⊛⊛
              endif;
            endif;
          endfor;
        else
          // hit ist bereits Nachfolgeraktivität des betrachten Knotens nPG.
          NFS := NFS ∪ FindActForNotificationPG(aADG, ADG, hit);              ⊛
        endif;
      case nPG ∈ A:
        if (( ah:= adg.IsDependent(aADG, nPG) == true) or
             (nPG.isDummyActivity == true)) then
          // nPG hat bereits ReleaseObjectVersion durchgeführt. → nicht informieren, da sonst Zyklus
          // bzw. nPG ist Dummy-Aktivität → ebenfalls nicht informieren
          NFS := NFS ∪ FindActForNotificationPG(aADG, ADG, nPG.Succ);         ⊛
        else
          NFS := NFS ∪ \{nPG\} ∪ nPG.ROAS;                                    ⊛⊛⊛
          // ROAS = Read-Only Activity Set: Menge aller lesender Aktivitäten, für die nPG auf diesem
          // Pfad die nächste Schreibende ist
        endif;
      case nPG == ⊢  or nPG == ᵗᵛ⊢ :
        NFS := NFS ∪ nPG.ROAS ∪ FindActForNotificationPG(aADG, ADG, nPG.Succ);  ⊛
      case nPG == ⟩◀ :
        NFS := NFS ∪ FindActForNotificationPG(aADG, ADG, nPG.Succ);           ⊛
      case nPG == ⟲◀ :
        // Überprüfe, ob erster Knoten in der Schleife bereits besucht wurde.
        if ( nPG.alreadyVisited == false ) then
          nPG.alreadyVisited := true                                          ⊛⊛
          NFS := NFS ∪ FindActForNotificationPG(aADG, ADG, nPG.Succ);         ⊛
        endif;
    endswitch;
  endif;
  return(NFS);
end;

begin // main
  // Phase 1: Bestimme die zu informierenden Aktivitäten, die auf zu aADG parallelen Zweigen liegen.
  // NFS = Notification Set
  NFS := FindActForNotificationPG(aADG, adg, nPG);
  // Phase 2: Ergänze diese Menge um die direkten Datenflussnachfolger.
  for ( aPAS ∈ aADG.potActSet^df_ROV(ov) ) do
```

```
    if (adg.IsDependent(aADG, aPAS) == false) then
      // Diese Aktivität aus potActSet^df_ROV hat noch kein ReleaseObjectVersion durchgeführt.
      NFS := NFS ∪ {aPAS}
    endif;
  endfor;
  return(NFS);
end;
```

#### 15.2.2.3 Korrektheit des Algorithmus *FindActForNotification*

**Terminierung**

Bei rekursiv definierten Algorithmen ist ihre Terminierung häufig nicht offensichtlich erkennbar. Die interne Methode *FindActForNotificationPG* durchläuft rekursiv alle Pfade von Parallelitätsgraphen. Erschwerend kommt bei diesem Algorithmus hinzu, dass er damit auf zyklischen Graphen operiert, womit unendlich lange Pfade und somit unendlich viele rekursive Aufrufe möglich sind.

Untersucht man den Algorithmus im Detail, so stellt man fest, dass jede Rekursion mit dem Nachfolger beziehungsweise den Nachfolgern des gerade bearbeitenden Knotens im Parallelitätsgraph gestartet wird (siehe mit ⊛ markierte Programmzeilen). Betrachtet man zunächst einmal schleifenfreie Parallelitätsgraphen, bei denen nur endlich lange Pfade möglich sind, so ist die Terminierung damit sicher. Die maximale Rekursionstiefe entspricht der Anzahl der Knoten im längsten Pfad durch den Parallelitätsgraph.

Schwieriger wird der Terminierungsnachweis, wenn man zyklische Parallelitätsgraphen vorfindet, da durch Schleifen unendlich lange Pfade möglich werden. Um dies im *FindActForNotificationPG*-Algorithmus zu vermeiden, wird jeder Schleifenbeginn (Symbol $\blacktriangleright^{\circlearrowleft}$, siehe mit ⊛⊛ markierte Programmzeile) beim ersten Besuch markiert. Wiederholte Durchläufe einer Schleife können damit erkannt und die Rekursion beendet werden. Dadurch ist auch bei Schleifen eine endliche Anzahl von *FindActForNotificationPG*-Aufrufen garantiert. Diese Vorgehensweise stellt auch sicher, dass durch vorzeitige Datenweitergabe keine Zyklen über Schleifen entstehen (vergleiche die in Abschnitt 15.1.3 skizzierte Problematik).

**Korrektheit der zurückgegebenen Aktivitätenmenge**

*FindActForNotificationPG* muss nicht nur terminieren, sondern auch gemäß den Anforderungen aus Abschnitt 15.1 die korrekte Menge an zu benachrichtigenden Aktivitäten liefern. Dabei wird bei den folgenden Betrachtungen mit *aADG* immer die Aktivität bezeichnet, welche die Weitergabe einer Objektversion angestoßen hat.

**Datenweitergabe zu parallelen Aktivitäten**

Existieren zu *aADG* keine parallelen Zweige, so ist der ihr zugeordnete Parallelitätsgraph leer und die Phase 1 im *FindActForNotification*-Algorithmus wird übersprungen.

Andernfalls wird rekursiv jeder Pfad des Parallelitätsgraphen mittels der Nachfolgerrelation durchlaufen. Die *switch*-Anweisung deckt bei den rekursiven Aufrufen alle möglichen Knotentypen eines

Parallelitätsgraphen ab. Existieren mehrere Nachfolger, so erfolgt für jeden dieser Nachfolger ein rekursiver Aufruf (Tiefensuche, siehe ⊛⊛⊛⊛ im Programmcode). Existieren keine Nachfolger, so terminiert *FindActForNotificationPG* und liefert eine leere Menge zurück. Die Rekursion endet für einen Pfad auch, wenn die erste Aktivität gefunden wurde, die noch kein $ReleaseObjectVersion$ durchgeführt hat (vergleiche mit ⊛⊛⊛ markierte Programmzeilen). Diese Aktivität muss zusammen mit allen nur lesenden Aktivitäten (Menge $ROAS$ [3]) über die neue Objektversion informiert werden. Sie wird folgerichtig durch den Algorithmus in die Menge *NFS* aufgenommen. Die Treffer aller rekursiven Aufrufe werden in der Menge *NFS* vereinigt.

Die Forderung den Datenaustausch zwischen bedingten Zweigen zu verhindern (vergleiche Abschnitt 15.1) bedingt eine besondere Vorgehensweise im Algorithmus. Er muss sicherstellen, dass Aktivitäten nicht über den „Umweg“ paralleler Zweige doch Daten mit Aktivitäten aus anderen bedingten Zweigen austauschen (vergleiche Abbildung 15.9).

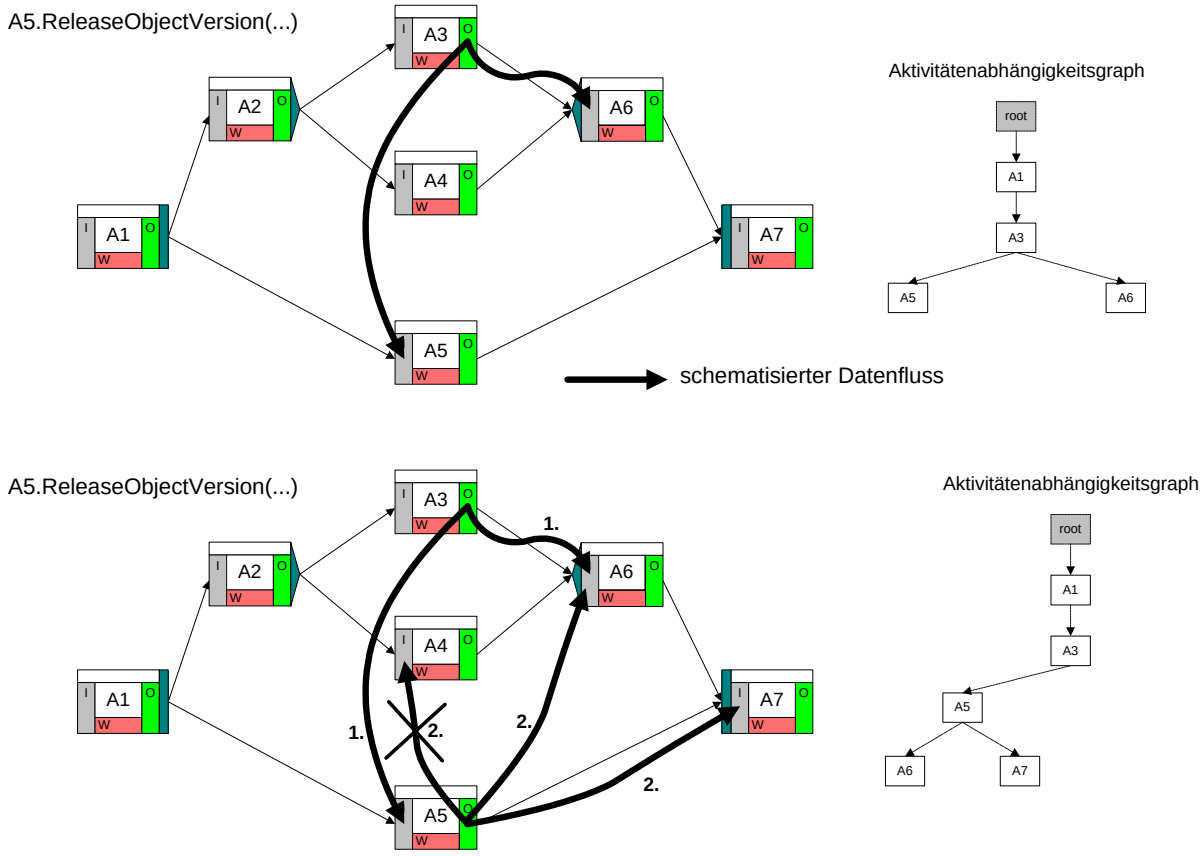



**Abbildung 15.9:** Schematisierter Datenfluss: Ein Datenfluss von Aktivität $A3$ zur Aktivität $A4$ in einem anderen bedingten Zweig darf auch nicht über den Umweg der zu $A3$ parallelen Aktivität $A5$ an $A4$ erfolgen.

Diese Forderung wird dadurch realisiert, dass *FindActForNotificationPG* überprüft, ob die Aktivität *aADG* bereits Daten aus einem der bedingten Zweigen erhalten hat. Die erste Aktivität dieses Zweigs ist in einem solchen Fall (vergleiche ⊛⊛⊛⊛⊛ im Programmcode) eine in direkter Linie verwandte

[3] Die Menge $A.ROAS$ (*Read-Only Activity Set*) enthält alle Aktivitäten eines Pfads, für den die Aktivität $A$ die nächste schreibende Aktivität repräsentiert. Vergleiche Umbau des Aktivitätenabhängigkeitsgraphen bei der Berücksichtigung von Datenflussabhängigkeiten in Abschnitt B.1.1.4.

Vorfahrin im Aktivitätenabhängigkeitsgraph. Sie wird als erste Aktivität dieses Zweiges dann in der Variablen *hit* gespeichert und es werden nur noch Aktivitäten aus diesem bedingten Zweig in die Treffermenge *NFS* aufgenommen. Diese Vorgehensweise schließt Datenaustausche wie in Abbildung 15.9 mit **X** markiert aus, da die anderen Zweige nicht weiter untersucht werden. Ihre Aktivitäten werden also nicht in die Treffermenge übernommen.

**Datenweitergabe an im Kontrollfluss nachfolgenden Aktivitäten**

Die Bestimmung der im Kontrollfluss nachfolgenden Aktivitäten, die entsprechend dem Datenfluss die weitergegebene Objektversion erhalten sollen, erfolgt in der Phase zwei. Die zur Aktivität *aADG* im Kontrollfluss als nächstes vom Objekt, zu dem die Objektversion $ov$ gehört, lesenden Aktivitäten konnten bereits zur Modellierungszeit bestimmt und in der Menge *adg.* $potActSet^{df}_{ROV}$ abgespeichert werden. Um zyklische Abhängigkeiten zu vermeiden, muss diese Menge – analog zur Phase eins – um solche Aktivitäten reduziert werden, die bereits an *aADG* eine Objektversion weitergereicht haben. Dies erfolgt wiederum mittels der Funktion *IsDependent*.

Die Gesamttreffermenge wird dann aus den gefundenen Mengen beider Phasen gebildet und enthält genau die zu benachrichtigenden Aktivitäten.

■

#### 15.2.2.4 Umbau des Aktivitätenabhängigkeitsgraphen

Der Algorithmus *ModifyADG* realisiert den dritten Schritt im $ReleaseObjectVersion$-Algorithmus. Er aktualisiert den Abhängigkeitsgraph, in dem alle durch den Algorithmus *FindActForNotification* gefundenen Aktivitäten als Kinder der weitergebenden Aktivität *aADG* in den Aktivitätenabhängigkeitsgraph eingefügt werden beziehungsweise umgehängt werden.

- Ist die Aktivität $Ai \in notifSet_{ROV}(A, op, oql)$ noch nicht im Aktivitätenabhängigkeitsgraph enthalten, dann füge $Ai$ als Kind von $A$ ein.[4]
- Andernfalls hänge $Ai$ zum Kind von $A$ um.

```
ModifyADG(aADG: A, adg: actDepGraph, nfs: ℘ (A )): actDepGraph
begin
 // Phase 3:
   for ( aNFS ∈ nfs) do
      if ( ahNFS:= adg.FindActivity(aNFS) == true) then
         // ahNFS ist bereits im Baum vorhanden  → umhängen
        adg.changeParentTo(ahNFS, aADG);
      else
         // Füge Aktivität aNFS als Kind von aADG ein.
        adg. insertasChildOfParent (ahNFS, aADG);
      endif;
   endfor;
   <Informiere alle Aktivitäten, die im ActDepGrph adg unter aADG hängen
    über die neue Objektversion ov (unter Berücksichtigung der Qualitätsstufen).>
end;
```

[4] Im Algorithmus *ModifyADG* wird der formale Parameter *nfs* mit $notifSet_{ROV}(A, op, oql)$ belegt.

Während bei einfach strukturierten **WEP**-Workflow-Graphen die zu benachrichtigenden Aktivitäten noch intuitiv zu bestimmen sind, ist dies bei komplexeren Workflow-Strukturen nicht mehr so einfach der Fall. Beispielsweise zeigt die Abbildung 15.10 verschiedene Zustände des Abhängigkeitsgraphen für den komplexeren Workflow aus Abbildung 15.6. Dabei wird davon ausgegangen, dass alle Aktivitäten vom gleichen Objekt lesen und schreiben.

### 15.2.3 Umstrukturierung des Aktivitätenabhängigkeitsgraphen bei $FinishActivity$

Das Beenden einer *unabhängigen* Aktivität hat Auswirkungen auf den Aktivitätenabhängigkeitsgraph eines Objekts. Da von einer endgültig beendeten Aktivität $A$ keine neue Objektversionen weitergegeben werden, kann diese aus den Abhängigkeitsgraphen der globalen Objekte entfernt werden, die mit ihren Ausgabeparametern verbunden sind. Befindet sich die Aktivität $A$ innerhalb einer Schleife, so kann sie allerdings bei einem weiteren Schleifendurchlauf erneut aktiviert werden. Sie muss in solchen Fällen über neue Objektversionen informiert werden. Bedingt durch die vorzeitige Datenweitergabe kann es dabei vorkommen, dass Vorgängeraktivitäten von $A$ bereits Objektversionen weitergegeben haben, die aber zur Vermeidung von Datenflusszyklen der gerade beendeten Aktivität $A$ vorenthalten wurden, weil diese Objektversionen direkt oder indirekt auf Daten basierten, die von der Aktivität $A$ weitergegeben wurden (vergleiche die in Abschnitt 15.1 geschilderter Problematik, insbesondere Abbildung 15.5). Dieses Informieren muss nun für die neue Inkarnation der Aktivität $A$ nachgereicht werden.

Um dies zur Laufzeit effizient gewährleisten zu können, müssen ähnlich zum $ReleaseObjectVersion$-Algorithmus nach Abschluss der Workflow-Modellierung die benötigten Informationen bei den betroffenen Aktivitäten zusammengefasst werden. Wie oben geschildert werden diese Informationen nur bei Aktivitäten benötigt, die sich innerhalb von Schleifen befinden. Bei jeder dieser Aktivitäten $A_{loop}$ wird deshalb für jeden ihrer Eingabeparameter $ip$ gespeichert:

- Die Menge der Aktivitäten, die gemäß des Kontrollflusses Vorgängeraktivitäten von $A_{loop}$ sind[5] und die für den jeweiligen Kontrollflusspfad als letztes in das globale Objekt im geforderten Qualitätsbereich schreiben, von dem $A_{loop}$ über den Eingabeparameter $ip$ liest.

- Die Menge der zu $A_{loop}$ parallelen Aktivitäten, die in das globale Objekt im geforderten Qualitätsbereich schreiben, von dem $A_{loop}$ liest. Diese Menge lässt sich leicht aus den bereits bestimmten Parallelitätsgraphen $parGraph^{cf}$ durch Analyse der Datenflüsse bestimmen. Dazu wird jede Aktivität aus dem Parallelitätsgraph $parGraph^{cf}(A_{loop})$ in die Menge aufgenommen, deren Ausgabeparameter mit dem betrachteten Eingabeparameter von $A_{loop}$ verbunden ist.

Beide Mengen werden zur Menge $A_{loop}.IPwriter(ip)$ vereinigt, die bereits in Kapitel 11 eingeführt wurde.

Zur Laufzeit soll eine Aktivität über ihre Eingabeparameter immer mit den für sie aktuellsten Objektversionen versorgt werden. Die für $A_{loop}$ aktuell gültigen Objektversionen lassen sich über den ReleaseMarker bestimmen:

[5] Die Vorgängerrelation $Pred_{cf}$ wird in analoger Weise zur Nachfolgerrelation $Succ_{cf}$ gebildet (siehe Kapitel 9).

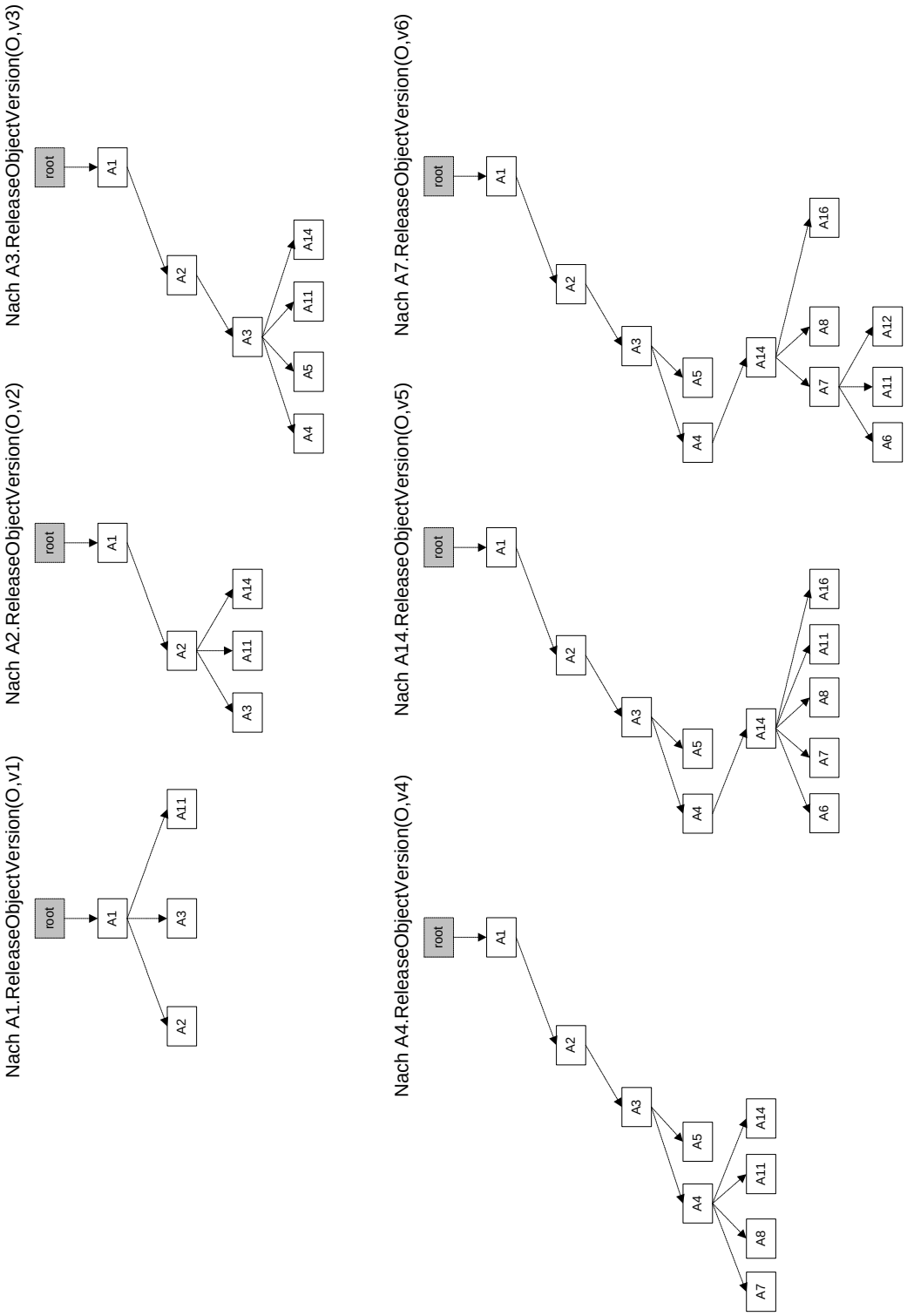



**Abbildung 15.10:** Aktivitätenabhängigkeitsgraph für den **WEP**-Workflow aus Abbildung 15.6

Für alle Objekte, die über Datenflusskanten mit einem Eingabeparameter von $A_{loop}$ verbunden sind

- suche den höchsten ReleaseMarker, der mit einer Aktivität aus $A_{loop}.IPwriter(ip)$ assoziiert ist. Dieser ReleaseMarker markiert die für die Aktivität $A_{loop}$ aktuell gültige Objektversion für den Eingabeparameter $ip$. Die dort enthaltene Aktivität wird im Folgenden mit $A_{current}$ bezeichnet. Falls kein ReleaseMarker gefunden wurde, so liegt noch keine Objektversion für diesen Eingabeparameter vor.
- Suche die Aktivität $A_{current}$ im entsprechenden Aktivitätenabhängigkeitsgraph.
- Lösche $A_{loop}$ aus dem Aktivitätenabhängigkeitsgraph. Die Kinder von $A_{loop}$ werden zu Kindern der Wurzel $root$.[6]
- Hänge die neue Inkarnation von $A_{loop}$ als Kind von $A_{current}$ in den Abhängigkeitsgraph und informiere $A_{loop}$ über die neue Objektversion.

### 15.2.4 Erweiterung auf mehrere globale Objekte

Die bisher skizzierten Algorithmen gehen nur von *einem* globalen Objekt aus. Da es im **WEP**-Modell selbstverständlich möglich ist, mehrere globale Objekte zu definieren und über diese den Datenfluss zu modellieren, können auch Datenflusszyklen über mehrere globale Objekte entstehen, die zu gegenseitig abhängigen Aktivitäten führen. Solche Zyklen können bei Datenabhängigkeiten zwischen parallelen Zweigen (siehe Abschnitt 15.2.4.1) und bei Schleifen (siehe Abschnitt 15.2.4.2) auftreten. Es wird nun gezeigt, welche Maßnahmen notwendig werden, um mittels obiger Algorithmen auch Datenflusszyklen über mehrere globale Objekte zu erkennen.

#### 15.2.4.1 Datenflusszyklen bei Parallelität

Am Beispiel in den Abbildungen 15.11 - 15.13 wird verdeutlicht, dass der $ReleaseObjectVersion$-Algorithmus in seiner bisherigen Form zu gegenseitig abhängigen Aktivitäten führen kann, wodurch möglicherweise durch wechselseitiges Zurücksetzen ein *Lifelock* entsteht.

Die Aktivität $A2$ gibt als erster Workflow-Schritt eine Objektversion des Objekts $O1$ weiter $(A2.ReleaseObjectVersion(O1, v2))$. $A3$ und $A4$ werden über die neue Objektversion informiert. $A3$ stößt ihre Konsistenzsicherung an. Sie ist dann abhängig von $A2$ (siehe Abbildungen 15.12 (a) und 15.12 (b)).

Nach Abschluss ihrer Konsistenzsicherungsphase (Abbildung 15.12 (c)) erzeugt $A3$ eine neue Objektversion von $O2$ (siehe Abbildung 15.12 (d) und gibt diese weiter (siehe Abbildung 15.13 (a)). Darüber wird $A2$ informiert. Bei einer angenommenen UndoRedo-Konsistenzsicherung stößt dann $A2$ ihre Konsistenzsicherung an. $A2$ ist dann abhängig von $A3$ bezüglich des Objekts $O2$ und $A3$ ist abhängig von $A2$ bezüglich des Objekts $O1$ (siehe Abbildung 15.13 (b)). Sie sind damit wechselseitig voneinander abhängig.

Bei der Konsistenzsicherung von $A2$ wird die bereits weitergegebene Objektversion $O1.v2$ zurückgezogen. Damit wird auch die von $A3$ davon abgeleitete Version $O1.v4$ ungültig. Es wird $A3$ informiert,

[6] Es werden nur unabhängige Aktivitäten aus dem Abhängigkeitsgraph gelöscht. Diese hängen immer direkt unter der Wurzel eines Abhängigkeitsgraphen.

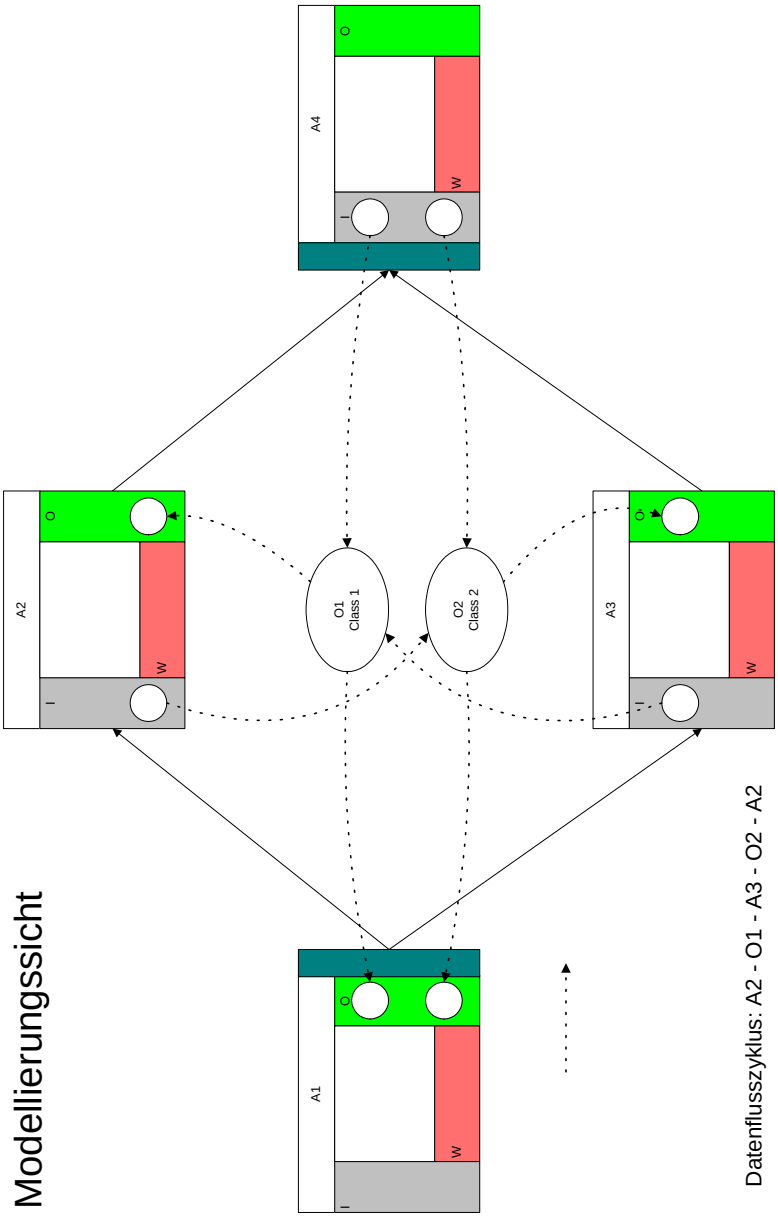
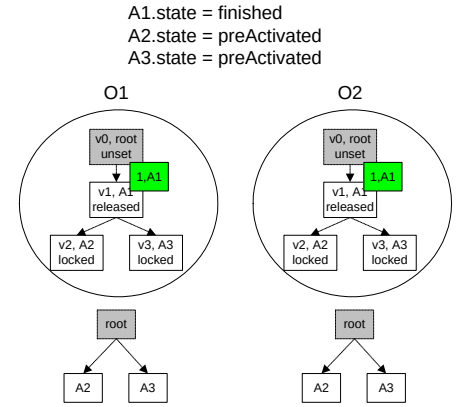
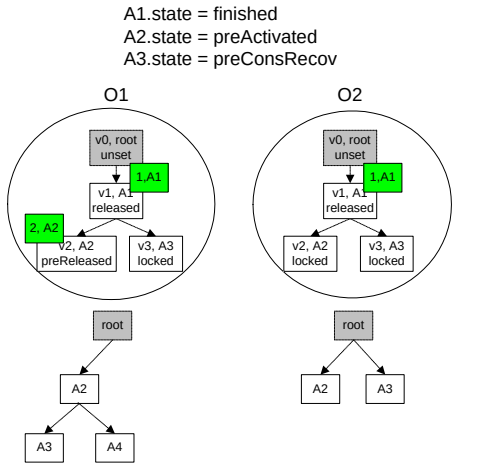


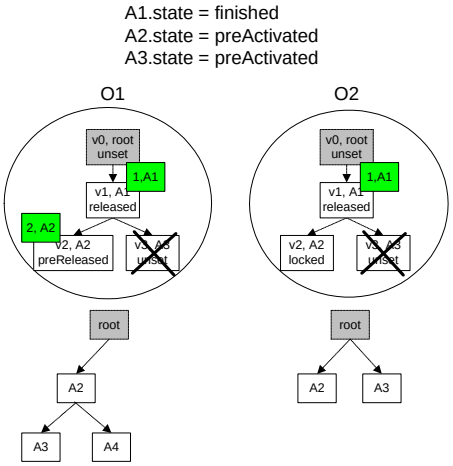

**Abbildung 15.11:** Datenflusszyklus über zwei Objekte bei parallelen Aktivitäten, Modellierungssicht



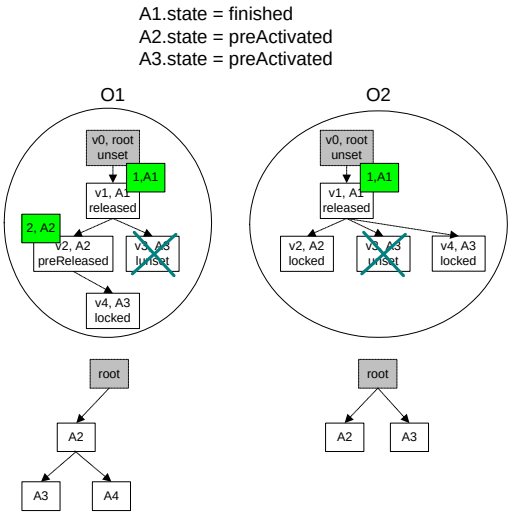

(a) Ausgangssituation




(b) Während $A2.ReleaseObjectVersion(O1, v2)$

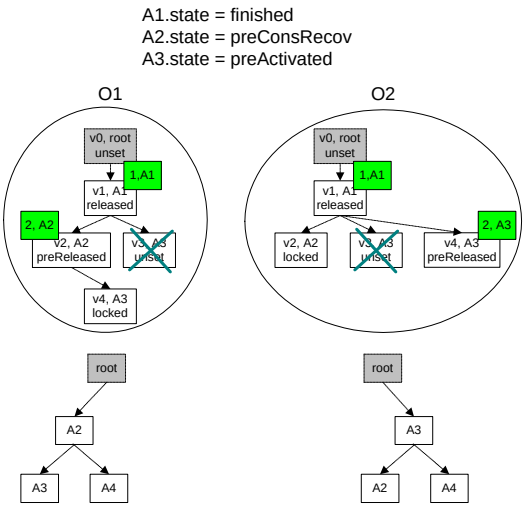



(c) Nach Abschluss UndoRedo-Konsistenzsicherung von $A3$

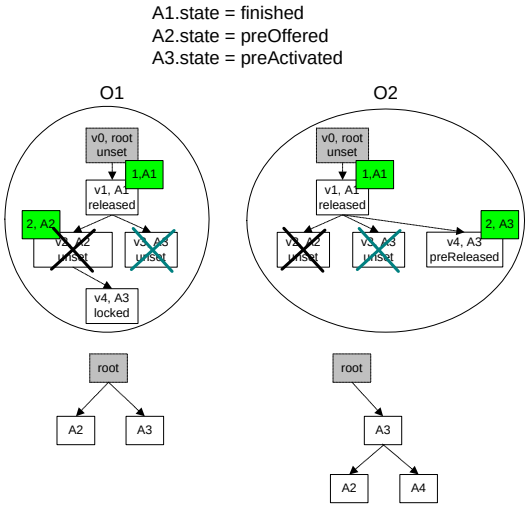



(d) Nach $A3.FetchObjectVersion(O1, v2)$

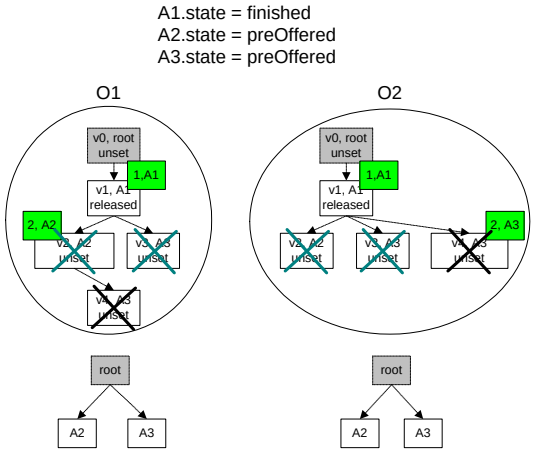
**Abbildung 15.12:** Datenflusszyklus über zwei Objekte bei parallelen Aktivitäten, Ausführungssicht, Teil 1




(a) Während $A3.ReleaseObjectVersion(O2, v4)$




(b) Nach Abschluss UndoRedo-Konsistenzsicherung von $A2$




(c) Nach Abschluss UndoRedo-Konsistenzsicherung von $A3$

**Abbildung 15.13:** Datenflusszyklus über zwei Objekte bei parallelen Aktivitäten, Ausführungssicht, Teil 2

weil nun wieder $O1.v1$ für sie gültig ist. $A3$ stößt ihre Konsistenzsicherung an und macht $O1.v4$ ungültig. Damit ist der Ausgangszustand wieder erreicht (siehe Abbildung 15.13 (c)) und der Lifelock kann wieder von vorn beginnen.

#### 15.2.4.2 Datenflusszyklen bei Schleifen

Transitive Datenabhängigkeiten existieren auch bei Schleifen. Dies wird durch die Abbildungen 15.14 - 15.15 illustriert. Auch hier ist die Ursache bei nicht synchronisierten Abhängigkeitsgraphen zu finden. Die vollständig durchlaufene Schleife (vergleiche Abschnitt 15.1.3) kann deshalb nicht anhand des Abhängigkeitsgraphen von $O1$ identifiziert werden, weshalb bei einer UndoRedo-Konsistenzsicherung erneut ein Lifelock entstehen würde.

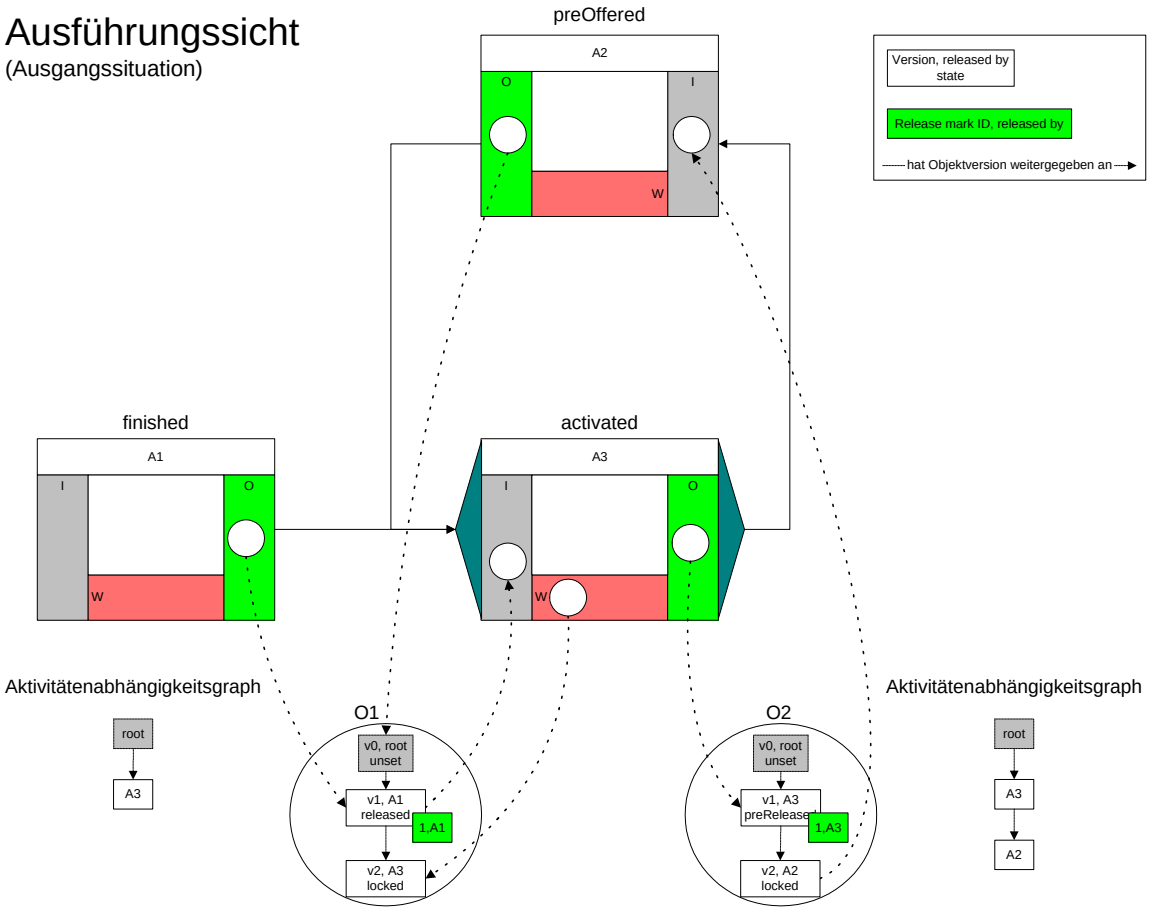



**Abbildung 15.14:** Datenflusszyklus über mehrere Objekte bei Schleifenaktivitäten, Ausgangssituation

#### 15.2.4.3 Behandlung von Datenflusszyklen

Gegenseitige Abhängigkeiten zwischen Aktivitäten entstehen dadurch, dass die Abhängigkeitsgraphen der beteiligten globalen Objekte unterschiedliche Reihenfolgen bei der Datenweitergabe dokumentieren. Um dies zu vermeiden, wird für alle in einem Datenflusszyklus enthaltenen Objekte ein

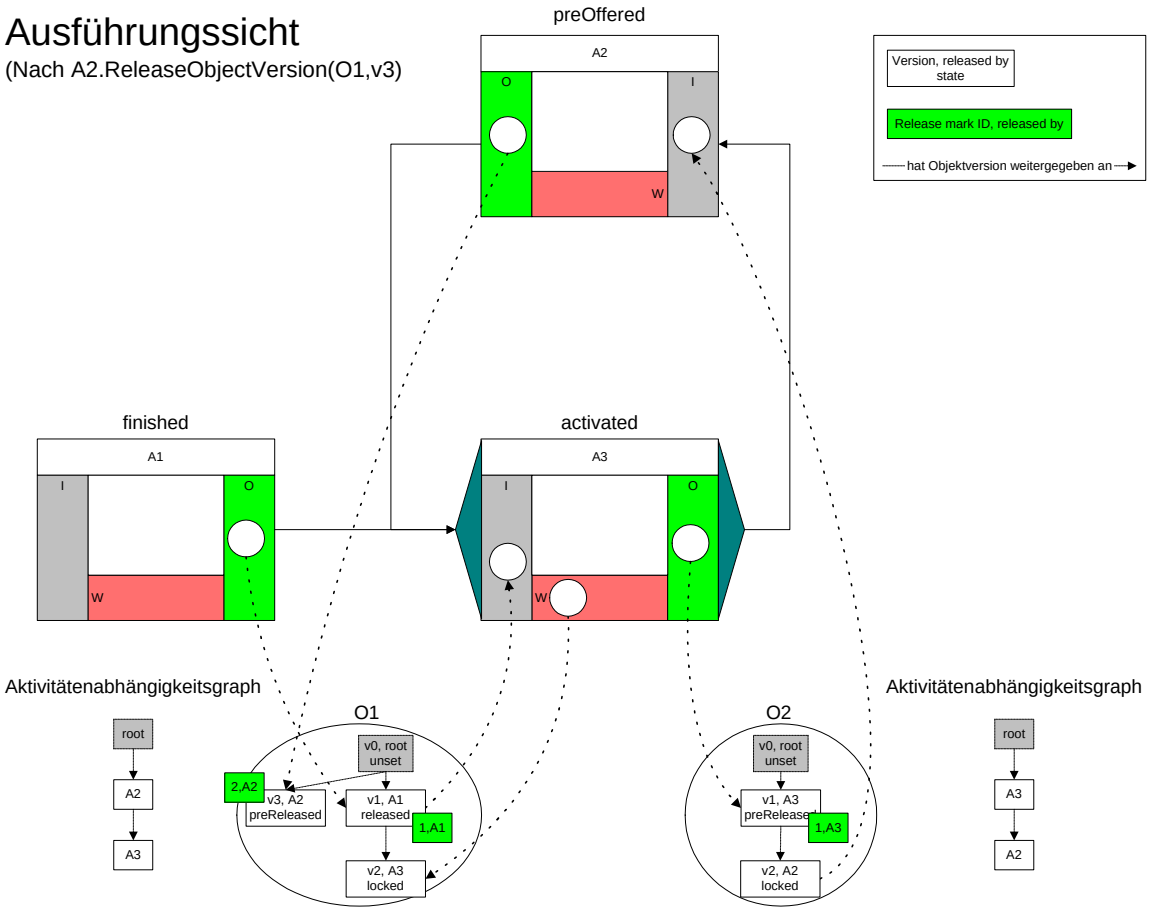



**Abbildung 15.15:** Datenflusszyklus über mehrere Objekte bei zwei Aktivitäten in einer Schleife, nach $A2.ReleaseObjectVersion(O1, v1)$

*gemeinsamer* Abhängigkeitsgraph verwaltet. Damit ist die gegenseitige Abhängigkeit von Aktivitäten aufgebrochen und die für *ein* globales Objekt entwickelten Algorithmen aus Abschnitt 15.2.2 können angewandt werden.

# Kapitel 16

# Durchführung der Traversierung zur Laufzeit

Es folgt nun die Beschreibung, wie das in den Abschnitten 7.2.5 und 8.3.1 eingeführte Modellierungskonstrukt der dynamischen Parallelität zur Laufzeit behandelt wird. Kernpunkt der Ausführungssemantik ist der im Folgenden beschriebene Traversierungsalgorithmus, der bestimmt, welche der dynamisch erzeugten Zweige neu erzeugt, gelöscht oder mit neuen Eingabeobjektversionen aktualisiert werden.

- Ein Zweig muss *neu erzeugt* werden, wenn bei der aktuellen Traversierung eine Subobjektversion als Treffer selektiert wurde, ohne dass die beim vorhergehenden $ReleaseObjectVersion$ weitergegebene Objektversion auch als Treffer gefunden wurde.
- Ein Zweig muss *gelöscht* und die in diesen Zweig gestarteten Aktivitäten müssen zurückgesetzt werden, wenn die diesem Zweig zugeordnete Subobjektversion beim vorhergehenden $ReleaseObjectVersion$ als Treffer gefunden wurde, bei der aktuellen Traversierung jedoch keine der nachfolgenden Objektversionen dieses Subobjekts als Treffer markiert wurde. Dies ist der Fall, wenn die mit dem ReleaseMarker der aktuellen $ReleaseObjectVersion$-Operation markierte Objektversion nicht den Bedingungen des Traversierungsmerkmals genügt oder keine der nachfolgenden Objektversionen dieses Subobjekts einen ReleaseMarker erhalten haben, weil sie für diese $ReleaseObjectVersion$-Operation nicht mehr relevant sind.
- Ein Zweig muss *aktualisiert* werden, wenn die Traversierung bei der vorhergehenden und der aktuellen $ReleaseObjectVersion$-Operation einen Treffer beim gleichen Subobjekt gefunden hat.

Es wird nun zunächst die skizzierte Arbeitsweise der Traversierung anschaulich in Abschnitt 16.1 illustriert. Der Abschnitt 16.2 beinhaltet den Traversierungsalgorithmus im Detail.

## 16.1 Die Arbeitsweise des Traversierungsalgorithmus

Ausgangspunkt der Illustration ist der in Abbildung 16.1 gezeigte mögliche Bearbeitungszustand eines **WEP**-Workflows mit dynamischer Parallelität. Die Aktivität $A1$ hat eine erste Objektversion der

Klasse $class1$ mit insgesamt drei über die Relation $rl1$ verknüpften Subobjekte der Klasse $class2$, die alle bei einer $ReleaseObjectVersion$-Operation in der Version $v1$ weitergegeben wurden (siehe ReleaseMarker $(1, A1)$ in Abbildung 16.1).

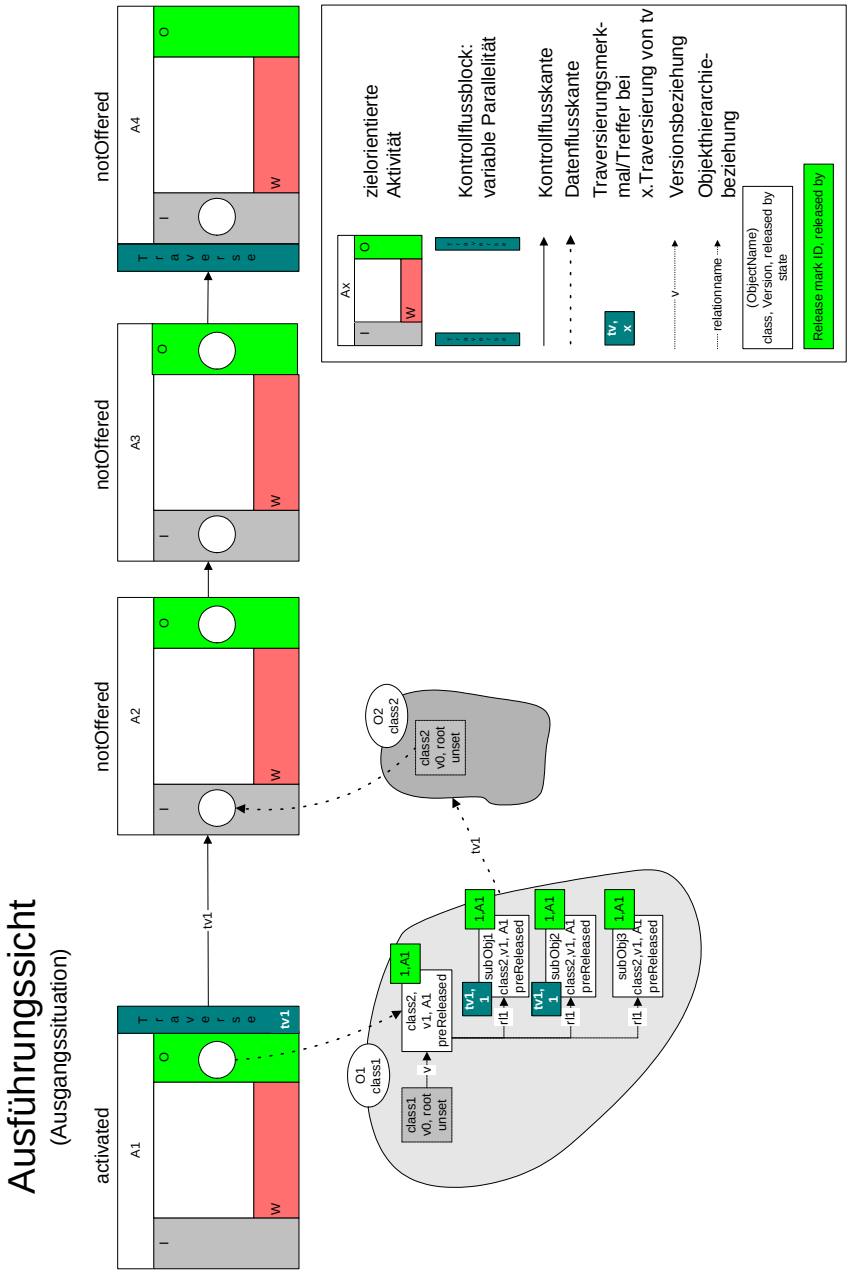



**Abbildung 16.1:** Arbeitsweise des Traversierungsalgorithmus, Ausgangssituation

Bei der Traversierung dieser komplexen Objektversion werden zwei ihrer Subobjekte als Treffer gefunden und mit dem Traversierungs-Marker $tv1$ markiert. Es werden deshalb zwei parallele Zweige sowie zwei globale Objekte $O2.1$ und $O2.2$ generiert, die über Datenflusseingangskanten den entsprechenden Aktivitäten zugewiesen werden (siehe Abbildung 16.2).

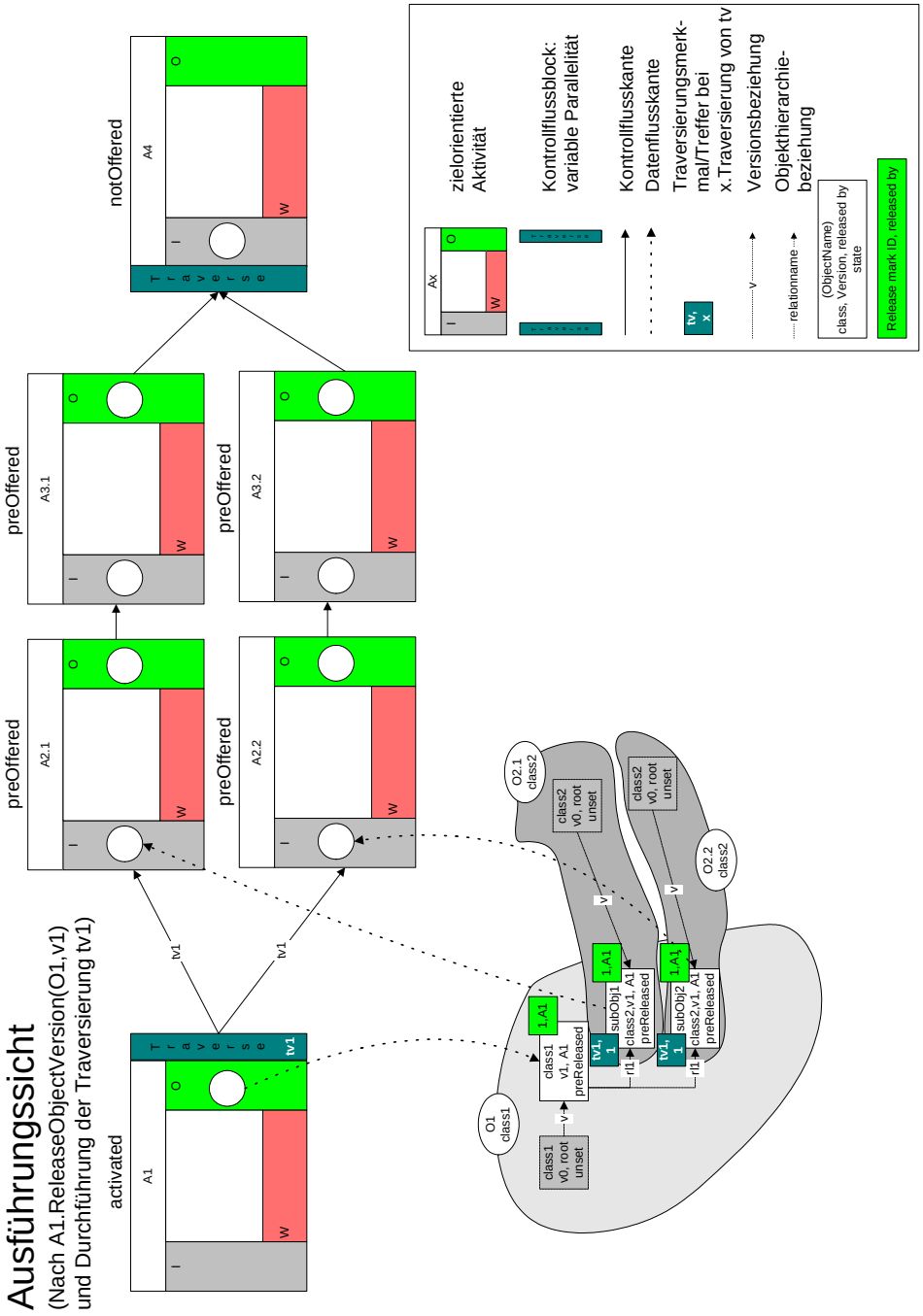



**Abbildung 16.2:** Arbeitsweise des Traversierungsalgorithmus nach der 1. Traversierung

In Abbildung 16.3 erzeugen die Aktivitäten $A1$ und $A2.2$ neue Objektversionen. Aktivität $A1$ gibt ihre erzeugte Objektversion, die auf das „alte“ Subobjekt *subObj1* und ein „neues“ Subobjekt *subObj3* verweist, in ihrer zweiten $ReleaseObjectVersion$-Operation weiter, so dass erneut eine Traversierung angestoßen werden muss. Abbildung 16.4 zeigt diese Situation vor Durchführung der Traversierung.

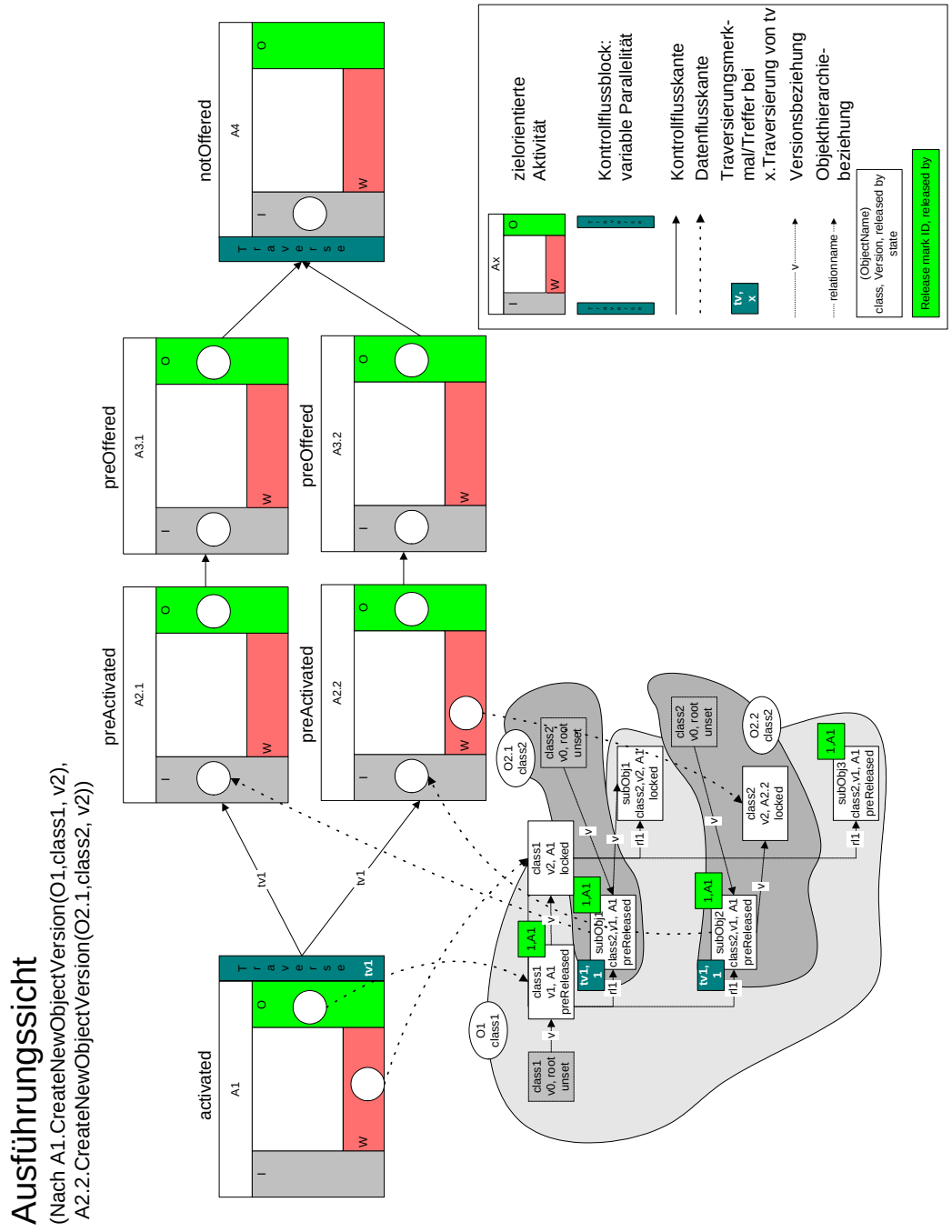



**Abbildung 16.3:** Arbeitsweise des Traversierungsalgorithmus nach Erzeugung neuer Objektversionen

Bei der zweiten Traversierung ergibt sich jedoch nur noch ein Traversierungstreffer, der auf eine Objektversion verweist, die von der Objektversion abgeleitet wurde, die dem globalen Objekt $O2.1$

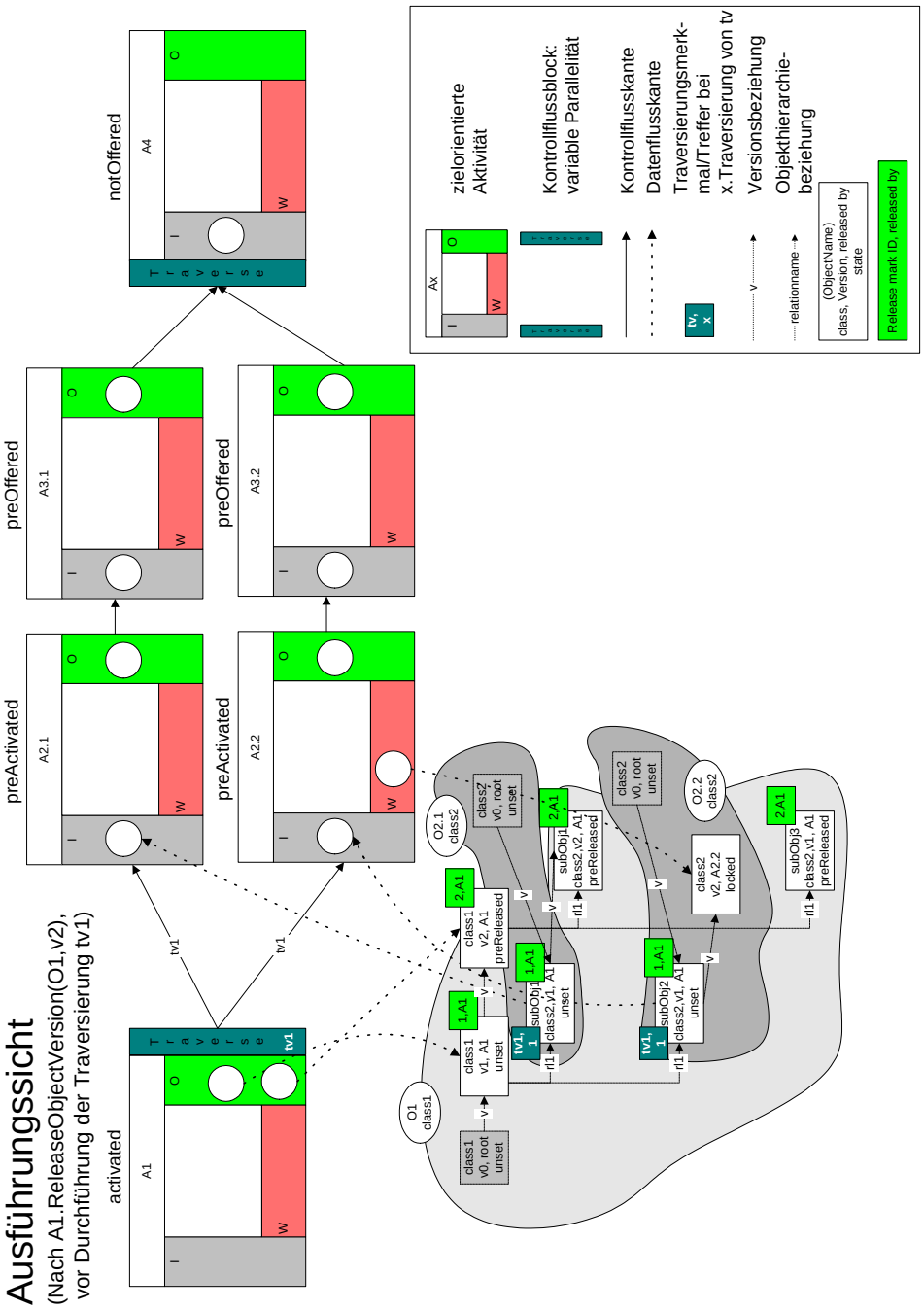



**Abbildung 16.4:** Arbeitsweise des Traversierungsalgorithmus nach der $ReleaseObjectVersion$-Operation und vor der 2. Traversierung

zugewiesen wurde. Diese Konstellation führt dazu, dass der in Abbildung 16.4 obere parallele Zweig über die neue Objektversion informiert werden muss (Aktualisierungsfall). Außerdem muss der untere Zweig gelöscht werden, da weder die bei der ersten Traversierung diesem Zweig zugewiesene Objektversion noch eine ihrer Nachfolgerversionen bei der aktuellen Traversierung als Traversierungstreffer markiert wurde. Abbildung 16.5 zeigt diese Aufräumungsarbeiten. In Abbildung 16.6 ist das Ergebnis nach Aktualisierung der Workflow-Struktur zu sehen.

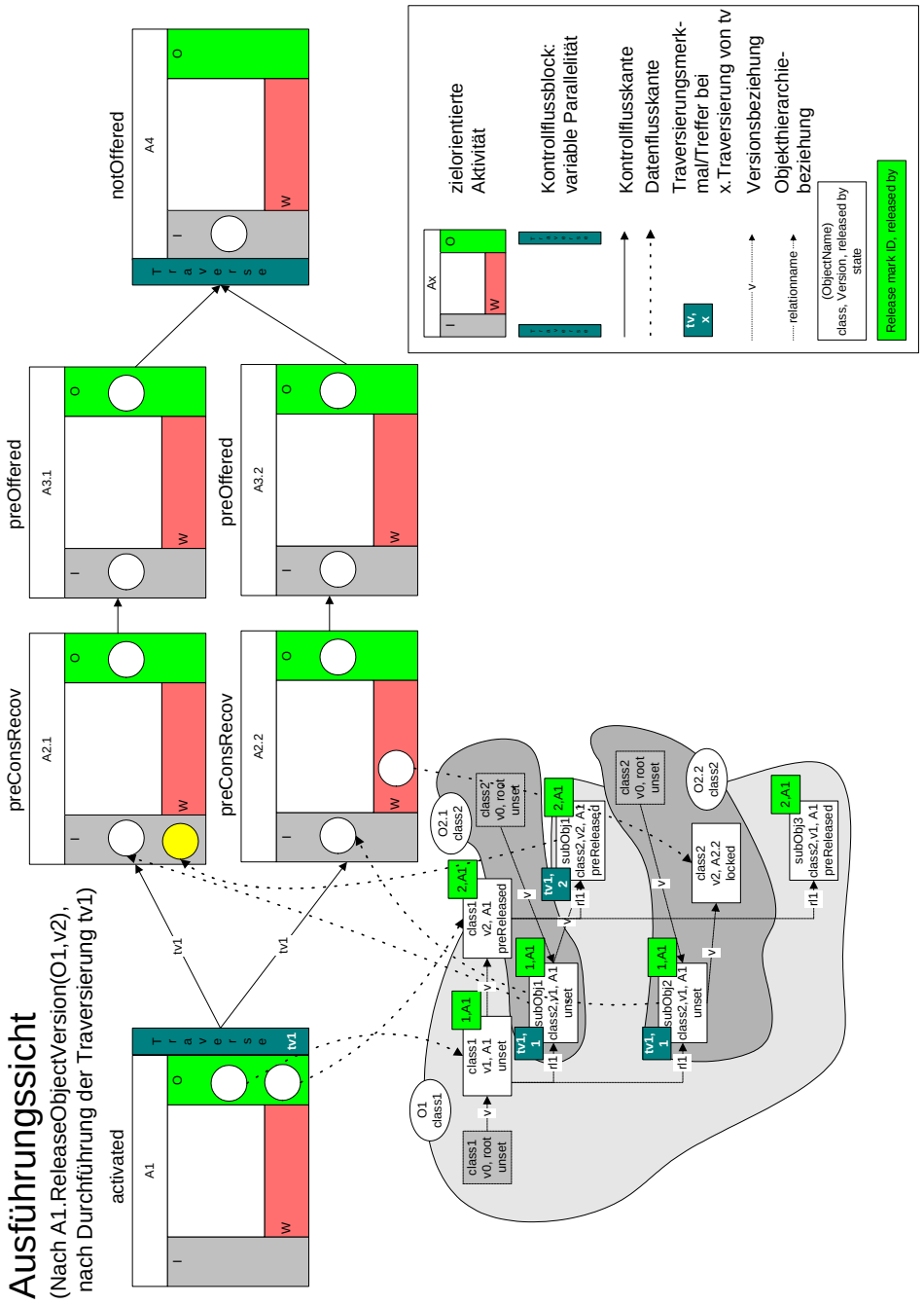



**Abbildung 16.5:** Arbeitsweise des Traversierungsalgorithmus während der Aufräumungsarbeiten

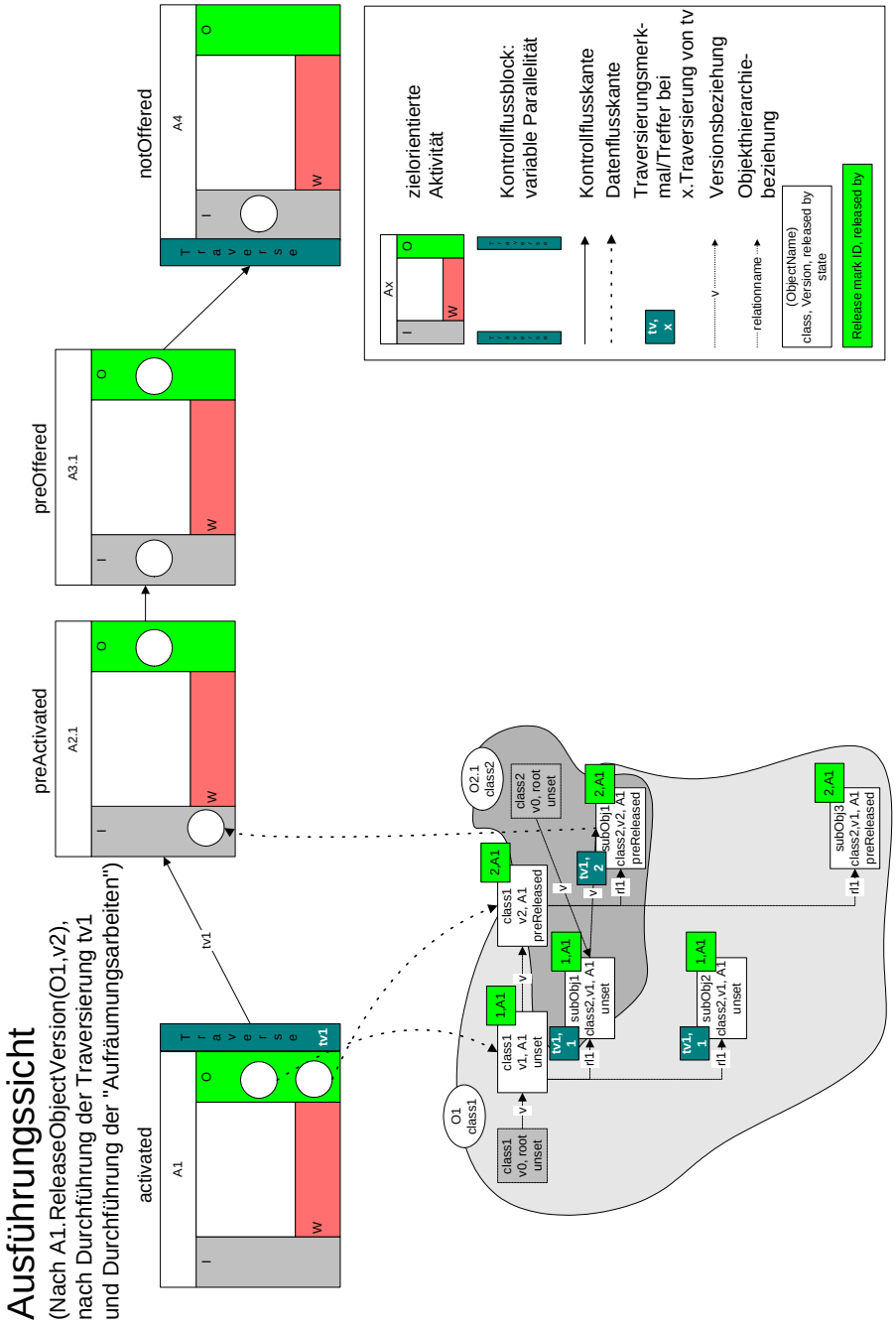



**Abbildung 16.6:** Arbeitsweise des Traversierungsalgorithmus nach Workflow-Struktur-Aktualisierung

## 16.2 Der Traversierungsalgorithmus

Der folgende Algorithmus spiegelt die drei in der Einleitung beschriebene Fälle, der *Aktualisierung* (siehe ⊛ im Algorithmus), der *Neuerzeugung* (siehe ⊛⊛ im Algorithmus) und der *Löschung* nicht mehr benötigter Zweige (siehe ⊛⊛⊛ im Algorithmus) wider.

Zur Identifikation der zu löschenden Zweige benötigt der Traversierungsalgorithmus die Menge der Zweige, die bei der letzten Traversierung dynamisch erzeugt wurden ($dppSet_{previous}$, siehe im Algorithmus ⊛⊛⊛⊛), und vergleicht diese am Schluss mit der Menge $dppSet_{current}$, welche die bei der aktuellen Traversierung erzeugten Zweige enthält. Alle Zweige, die nur in der $dppSet_{previous}$-Menge vorkommen, können zusammen mit ihren assoziierten globalen Objekten gelöscht werden.

Die folgenden Definitionen werden im Algorithmus verwendet:

- *dynParPathSet(releaseMarker, traverseFeature)* beschreibt die Menge aller Pfade innerhalb des durch *traverseFeature* referenzierten dynamischen Parallelitätskonstrukts.
- *ovHitsOfTraverse(object, releaseMarker, traverseFeature)* bezeichnet die Menge aller Treffer zum gegebenen Objekt *object*, ReleaseMarker *releaseMarker* und Traversierungsmerkmal *traverseFeature*, also die Menge der sich qualifizierten Objektversionen.

Für ein gegebenes Objekt *object*, einen gegebenen ReleaseMarker *releaseMarker* und ein gegebenes Traversierungsmerkmal *traverseFeature* lautet der Traversierungsalgorithmus:

*// Menge der variablen parallelen Pfade*
$dppSet_{previous}$ := *dynParPathSet(releaseMarker.GetPreviousReleaseMarker, traverseFeature )* ⊛⊛⊛⊛
$dppSet_{current}$ := ∅
*// Führe die Traversierung auf der aktuell weitergegebenen Objektversion durch.*
***for*** *( ovhit ∈ ovHitsOfTraverse(object , releaseMarker , traverseFeature )) **do***
  ***if*** *(∃ $dov_{previous}$ ∈ ovhit.Predecessors, ∃ $dpp_{previous}$ ∈ $dppSet_{previous}$ :*
    *$dov_{previous}$.ReleaseMarker == (ovhit.ReleaseMarker).GetPreviousReleaseMarker*
    ***and***
    *$dov_{previous}$ == $dpp_{previous}$.GetAttachedTraverseHit)*
  ***then*** *// Aktivierungsfall* ⊛
    *// Es existiert in der Objektversionshierarchie des Subobjekts, zu dem der jetzige Traversierungstreffer ovhit*
    *// gehört, bereits eine Vorgängerobjektversion, die Treffer bei der vorangegangenenTraversierung war.*
    *// Diese Vorgängerobjektversion wurde dem dynamischen Pfad $dpp_{previous}$ zugeordnet.*
    *// Dieser Pfad erhält nun die aktuellere Objektversion zugewiesen.*
    *$dpp_{previous}$. SetAttachedTraverseHit ( ovhit )*
    *// Übernehme diesen Pfad in die Menge der aktuellen dynamischen Pfade.*
    *$dppSet_{current}$:= $dppSet_{current}$ ∪ $dpp_{previous}$*
    *$dppSet_{previous}$:= $dppSet_{previous}$ \ $dpp_{previous}$*
    *<Sende UpdateRequest über die entsprechende Datenflusskante an die erste Aktivität des betroffenen Pfades. >*
  ***else*** *// Neuerzeugungsfall* ⊛⊛
    *// Kein Treffer bei der vorangegangenen Traversierung für dieses Subobjekt*
    *// → Erzeuge neuen dynamischen Pfad und ordne ovhit diesem zu.*
    *$dpp_{new}$ := dynParBlock.CreateNewVarPath*
    *$dpp_{new}$ . SetAttachedTraverseHit ( ovhit )*
    *$dppSet_{current}$:= $dppSet_{current}$ ∪ $dpp_{new}$*
  ***endif***;
***endfor***;
*// Löschfall* ⊛⊛⊛
*// Lösche die übriggebliebenen Pfade aus $dppSet_{previous}$.*
*< Sende UndoRequest an die jeweils erste Aktivität aus jedem Pfad aus $dppSet_{previous}$ und lösche nach Durchführung des UndoRequests die den Pfaden zugeordneten globalen Objekte.>*

# Kapitel 17

# Ad-hoc-Interaktionsformen während Simultaneous-Engineering-Phasen

## 17.1 Aufgaben von Konsolidierungsrunden

Herkömmliche Workflow-Management-Systeme können nur verlaufsorientierte asynchrone Kooperationen zwischen den Prozessbeteiligten unterstützen. In der Praxis arbeiten die in einem Prozess involvierten Personen nicht nur in solchen asynchronen und vorgegebenen Kooperationsformen zusammen. Vielmehr treten je nach Ort- und Zeitverhältnisse weitere Kooperationsformen auf, die in der Regel aus einem situationsbedingten Abstimmungsbedarf resultieren. Aus diesem Grund fordern viele Anwender auch IT-Unterstützung für spontane synchrone Kooperationsformen. Diese Arbeitsformen werden umso wichtiger je mehr Simultaneous-Engineering in den Prozessen gefordert wird, da sich die Zahl der gleichzeitig arbeitenden Personen dadurch erhöht und somit auch ein größerer Abstimmungsbedarf besteht. Es ist damit offensichtlich, dass solchen synchronen Gruppenarbeitsmechanismen im **WEP**-Workflow-Management-System eine besondere Bedeutung zukommt (vergleiche auch die Anforderungen aus Teil I der Arbeit). Diesen Anforderungen trägt das **WEP**-Workflow-Management-System durch das Konzept der *Konsolidierungsrunden* Rechnung.

## 17.2 Anforderungen

Die folgenden Anforderungen [Kno99, KFW01] müssen **WEP**-Konsolidierungsrunden umsetzen:

*Reduzierung des Abstimmungsaufwands:*
: Konsolidierungsrunden sollen zeitaufwändige persönliche Abstimmungstreffen simultan und (global) verteilt arbeitender Prozessbeteiligter reduzieren, um damit auch zur Minimierung der Gesamtlaufzeiten von Produktentwicklungsprozessen beizutragen. Für eine hohe Akzeptanz darf deshalb die Einberufung und Durchführung einer Konsolidierungsrunde nur geringen Zusatzaufwand für die Beteiligten verursachen. Eine Konsolidierungsrunde muss deshalb eng in den Workflow-Kontext eingebunden sein.

*Unterstützung verschiedener Varianten von Ad-hoc-Abstimmungen:*
Eine virtuelle Abstimmungsrunde muss entsprechend dem zu lösenden Problem konfigurierbar sein. Mögliche Konfigurationsparameter umfassen:

*Art des Kommunikationsmediums:*
Hier variieren die Anforderungen zwischen Unterstützung bilateraler Telefongespräche über asynchronen Informationsaustausch (Email, elektronische Umlaufmappen) bis zu IT-unterstützten Videokonferenzen mit mehreren Teilnehmern und gemeinsamen Zugriff auf benötigte Daten sowie Anwendungen (*shared applications*).

*Zeitpunkt der Durchführung:*
Neben der sofortigen Einberufung einer Abstimmungsrunde werden auch eine automatische Terminfindung und damit die Durchführung der Abstimmungsrunde zum gefundenen späteren Termin sowie eine völlig asynchrone Abstimmung gefordert.

*Art der Beschlussfassung:*
Gefordert wird hier entweder ein einstimmiger oder ein Mehrheits-Beschluss, der neben den erzeugten Benutzerdaten das erzielte Ergebnis einer Konsolidierungsrunde dokumentiert und an alle Teilnehmer weiterleitet.

*Zeitliche Überwachung:*
Da ein **WEP**-Workflow Meilensteinrestriktionen unterliegt, müssen die verschiedenen Phasen von **WEP**-Konsolidierungsrunden ebenfalls zeitlich begrenzt und überwacht werden.

## 17.3 Prinzipieller Ablauf einer Konsolidierungsrunde

Die beschriebenen Anforderungen führen zu dem in Abbildung 17.1 skizzierten prinzipiellen Ablauf einer Konsolidierungsrunde, die sich in drei Phasen untergliedern lässt:

Jede Konsolidierungsrunde beginnt mit der *Vorbereitungsphase*. Diese Phase wird durch einen beliebigen Aktivitätenbearbeiter ausgelöst, wenn dieser Abstimmungsbedarf bezüglich einer für ihn sichtbaren Objektversion benötigt. Durch Analyse des Status der Kontroll- und Datenflusskanten sowie der Aktivitäten bestimmt das **WEP**-Workflow-Management-System, für welche Workflow-Bearbeiter die Objektversion sichtbar ist und *schlägt* diese dem Initiator *als mögliche Teilnehmer vor*. Diese Teilnehmergruppe kann vom Initiator der Abstimmungsrunde ergänzt oder reduziert werden. Außerdem bestimmt der Initiator die Form der Abstimmungsrunde. Damit endet die Vorbereitungsphase einer Konsolidierungsrunde.

In der anschließenden *Konsolidierungsphase informiert das Workflow-Management-System* die ausgewählten Teilnehmer *automatisch* und stellt ihnen die benötigten Daten und Applikationen zur Verfügung. Damit beginnt die eigentliche Gruppenarbeit. Während der Gruppenarbeit, die in vielfältiger Form realisiert werden kann (siehe Abbildung 17.1 und [Kno99]), wird über die eingebrachte Objektversion diskutiert und verschiedene Lösungsvarianten entworfen, um einen für alle tragbaren Kompromiss zu erzielen.

Jede Konsolidierungsrunde endet mit der *Abstimmungsphase*, in der über das erreichte Ergebnis abgestimmt wird. Mit der *automatischen Übermittlung des Abstimmungsergebnisses* endet eine **WEP**-Konsolidierungsrunde.

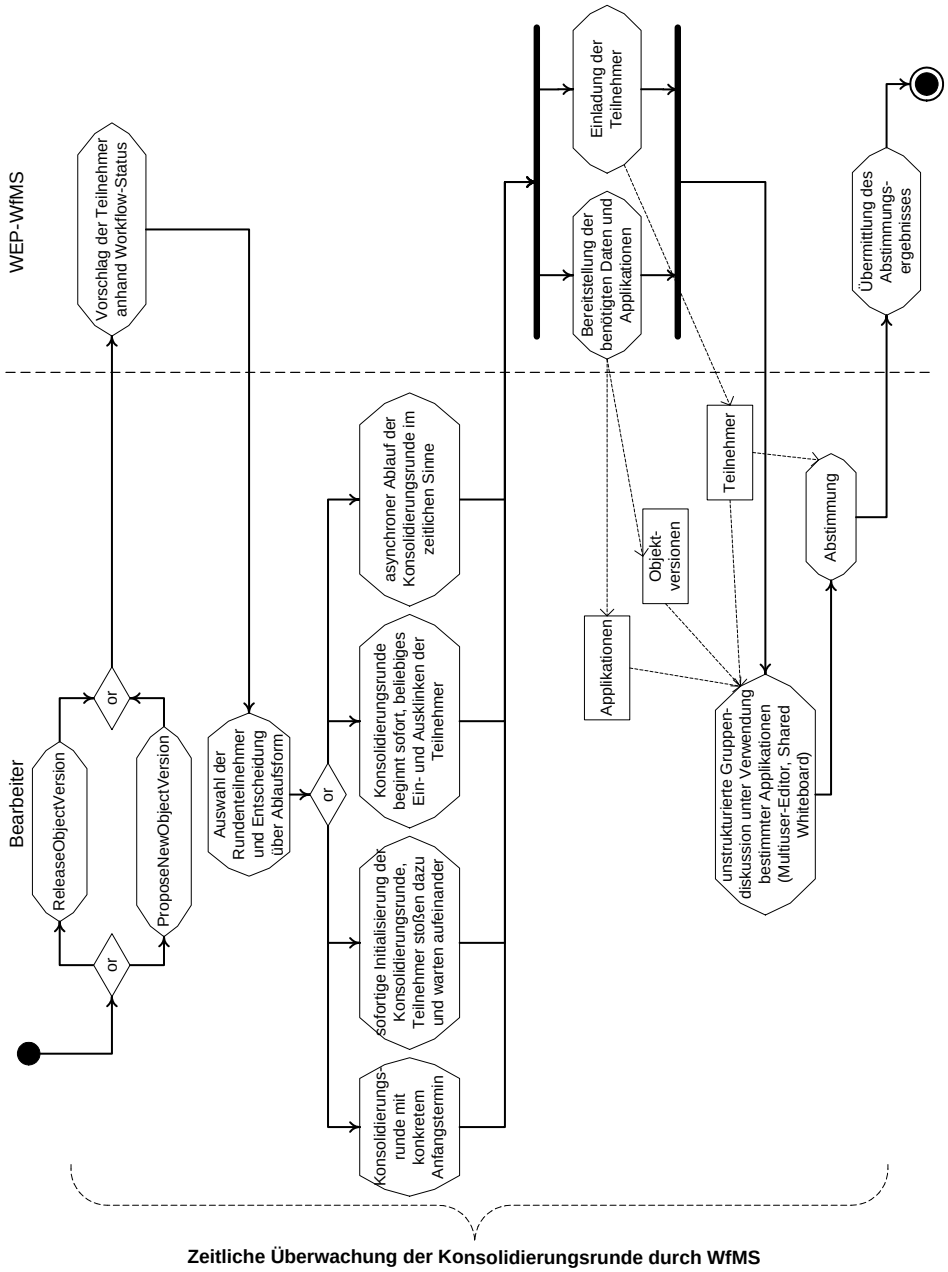



**Abbildung 17.1:** Prinzipieller Ablauf einer Konsolidierungsrunde

Während allen Phasen übernimmt die **WEP**-Workflow-Engine die *zeitliche Überwachung* einer **WEP**-Konsolidierungsrunde und leitet gegebenenfalls die nächste Phase ein.

Der prinzipielle Ablauf einer **WEP**-Konsolidierungsrunde zeigt bereits, dass viele administrative Schritte und die zeitliche Überwachung durch eine enge Kopplung mit dem **WEP**-Workflow-Management-System automatisiert werden können. Es wird auch deutlich, dass eine große Bandbreite an möglichen Abstimmungsszenarien notwendig ist, um alle relevanten Abstimmungsvarianten realisieren zu können.

## 17.4 Realisierung von WEP-Konsolidierungsrunden

Für ungeplante Kooperationsformen wurden bereits viele sogenannte *Gruppenarbeitskonzepte* entwickelt [DCSCW98, TSMB95, SB96, EGR91], sodass der Entwurf eines weiteren Gruppenarbeitsmodells zur Realisierung von **WEP**-Konsolidierungsrunden nicht zielführend wäre. Es gilt vielmehr die vorhandenen Ansätze auf ihre Tauglichkeit zur Umsetzung der Konsolidierungsrunden und ihre Integrationsfähigkeit in das **WEP**-Workflow-Management-System zu untersuchen. Dazu wurden im Rahmen einer begleitenden Diplomarbeit [Kno99] praxisrelevante Szenarien von Abstimmungsrunden detailliert und verschiedene Gruppenarbeitskonzepte bezüglich der Umsetzbarkeit der Szenarien untersucht, verglichen und kategorisiert. Bei der Auswahl der Gruppenarbeitsmodelle [SC94, PSS97, Par97, PS97, SPF98, KMS$^+$95, WPSH$^+$96, WPH$^+$97] wurde dabei versucht, sich auf solche Modelle zu konzentrieren, die bereits eine große Bandbreite von Kooperations- und Koordinationsmöglichkeiten abdecken.

Es stellte sich dabei heraus, das keine der untersuchten Konzepte *alle* entwickelten Szenarien befriedigend unterstützen konnte [Kno99]. Je nach anvisiertem Anwendungsgebiet haben alle betrachteten Systeme unterschiedliche Stärken und Schwächen in Bezug auf die Anforderungen der **WEP**-Konsolidierungsrunden. Systeme mit Schwerpunkt auf der Kooperation [PSS97, Par97, SC94] bieten meist keine Unterstützung für asynchrone Abstimmungsszenarien. Häufig fehlen auch Mechanismen für eine automatisierte Terminvereinbarung [SC94, KMS$^+$95, PS97, SPF98], eine zeitliche Überwachung des gesamten Ablaufs einer Konsolidierungsrunde [PSS97, Par97, SC94, WPSH$^+$96, WPH$^+$97] und Möglichkeiten zur Durchführung der Abstimmungsphase [SC94, KMS$^+$95, PS97, SPF98, WPSH$^+$96, WPH$^+$97].

Es bleibt jedoch festzuhalten, dass durch geschickte Kombination der betrachteten Gruppenarbeitskonzepte alle Anforderungen der **WEP**-Konsolidierungsrunden abgedeckt werden können. Die Entwicklung neuartiger Gruppenarbeitsmodelle für das **WEP**-Workflow-Management-System ist also nicht notwendig. Um die Untersuchungsergebnisse hier nicht detailliert wiederholen zu müssen, sei der interessierte Leser auf die Diplomarbeit [Kno99] verwiesen, in der auch verschiedene Realisierungs- und Erweiterungsmöglichkeiten diskutiert werden.

# Teil IV

# Diskussion verwandter Ansätze

# Kapitel 18

# Einführung

In der verfügbaren wissenschaftlichen Literatur finden sich bisher keine Ansätze, mit denen *alle* in Teil I der Arbeit skizzierten Anforderungen adäquat umgesetzt werden können. Dies ergibt sich einerseits daraus, dass die meisten vorgeschlagenen Konzepte für andere Anwendungsgebiete wie beispielsweise dem Software-Engineering entwickelt wurden, die partiell andere Anforderungen aufweisen. Anderseits ist es auch in der Komplexität der Praxisanforderungen begründet. Viele Lösungen konzentrieren sich deshalb auf Teilaspekte der Fragestellungen aus Teil I, was natürlich ihre praktische Einsetzbarkeit der wissenschaftlich sicherlich sehr interessanten Lösungsansätze für die hier anvisierte Produktentwicklung einschränkt.

Die meisten Lösungsansätze stellen sich dabei den prozessorientierten Fragestellungen aus Teil I. Die prozessorientierten Modelle nehmen deshalb auch den größten Raum der im Folgenden beschriebenen verwandten Arbeiten ein. Für eine adäquate Gesamtlösung muss jedoch die Prozesssteuerung im engen Zusammenspiel mit dem Daten- und Projektmanagement gesehen werden. Ersteres ergibt sich aus der Tatsache, dass die Prozesssteuerung Konzepte zur Weitergabe vorläufiger Objektversionen für das Simultaneous-Engineering bereitstellen muss. Kenntnisse über die komplexe Struktur und Qualität der Datenobjekte ermöglichen natürlich eine feingranularere und gezieltere Datenbereitstellung. Bei der Diskussion verwandter Ansätze können deshalb wissenschaftliche Lösungen aus dem Umfeld des Produktdaten- und Konfigurationsmanagements nicht außer Acht gelassen werden. Aus diesem Umfeld werden Modelle aus der Literatur betrachtet, die auch Prozesssteuerungskonzepte beinhalten.

Lösungsansätze zur Projektplanung- und -steuerung sind im Kontext dieser Arbeit aus zwei Gründen interessant zu diskutieren: Erstens wird eine zeitliche Führung und Überwachung unstrukturierter Teilprozesse benötigt, wo der Einsatz dieser Ansätze hilfreich sein kann. Zweitens sind die hier betrachteten operativen Produktentwicklungsprozesse in der Praxis in einen übergeordneten Projektmeilensteinplan eingeordnet, dessen Vorgaben berücksichtigt werden müssen. Im Wesentlichen werden hier nur solche Planungsmodelle diskutiert, die Planungs- und operative Steuerungsaspekte miteinander verschränken.

Der Vergleich verwandter Ansätze gliedert sich deshalb in drei Hauptkapitel: Kapitel 19 widmet sich prozessorientierten Ansätzen, Kapitel 20 stellt Verfahren zum Management komplex strukturierter Daten vor und in Kapitel 21 werden Modelle zur Steuerung langdauernder Projekte diskutiert. Neben einer allgemeinen Diskussion werden auch vielversprechende Ansätze detaillierter betrachtet. Im Kapitel 22 werden die Diskussionsergebnisse der wichtigsten wissenschaftlichen Ansätze tabellarisch

zusammengefasst. Das Kapitel 23 zeigt kurz die Defizite kommerzieller Systeme bei der Umsetzung der Praxisanforderungen aus Teil I auf.

# Kapitel 19

# Vergleich mit Ansätzen aus dem Prozessmanagement

Prozessorientierte Ansätze stellen immer die zu bearbeitende Aufgabe in den Mittelpunkt der Betrachtung. Ein Prozess wird modelliert, indem eine komplexe Aufgabe (rekursiv) in einfachere Subaufgaben aufgegliedert wird und zwischen den Aufgaben Abhängigkeiten definiert werden. Abhängigkeiten können unterschiedlicher Natur sein: Neben der Komposition und Dekomposition sind hier insbesondere Daten- und Kausalabhängigkeiten sowie zeitliche Restriktionen zu nennen [JB96].

Prozessorientierte Ansätze lassen sich anhand ihres verwendeten Formalismus kategorisieren. Häufig werden dazu gerichtete Graphen verwendet [Jab94, LA94, DKR+95, MHR96]. Als größter Vertreter dieser Kategorie sind hier sicherlich Petrinetze unterschiedlicher Mächtigkeit zu nennen [SR93, Gru93, MMJL95, Obe96, HOV93, MS93]. Aber auch State- und Activity-Charts [WDM+95, Tee96], regelbasierte [KYH95, RW92] und graphgrammatik-basierte [HJKW97] Metamodelle sind zu finden.

Im Folgenden werden verschiedene prozessorientierte Ansätze, die interessante Konzepte zur Realisierung der Anforderungen aus Teil I besitzen, vorgestellt, mit den **WEP**-Konzepten verglichen und auf ihre Eignung für Produktentwicklungsprozesse detaillierter untersucht.

## 19.1 Graphgrammatik-basierte Systeme, AHEAD

Graphgrammatiken und -ersetzungsregeln werden bisher eher selten zur Beschreibung von Workflows und ihrer Ausführungslogik verwendet [HJKW95, BK95]. Es wird deshalb auf eine allgemeine Bewertung graphgrammatik-basierter Konzepte verzichtet und stattdessen gleich auf den einzigen Ansatz dieser Kategorie, dem **AHEAD**-Modell, detaillierter eingegangen. Der **AHEAD**-Ansatz wurde zur Lösung von Anforderungen konzipiert, die sehr viel Ähnlichkeiten zu den Anforderungen aus Teil I aufweisen. Dabei wird auch die Idee und Verwendung von Graphgrammatiken sowie Graphersetzungsregeln vorgestellt.

### 19.1.1 Zielsetzung des AHEAD-Projekts

Primäres Ziel des **AHEAD**-Ansatzes (Adaptable and Human-Centered Environment for the Management of Development Processes) [Wes01] ist die rechnergestützte Koordination von Software-Entwicklungsprozessen.

Charakteristisch für Software-Entwicklungsprozesse ist ihre Dynamik, wodurch sich konkrete Aufgaben und Prozessstruktur erst zur Laufzeit ergeben. Komplexe Software-Entwicklungsprozesse können deshalb nur selten im Voraus vollständig geplant werden. Zwar stehen grundlegende Aufgabentypen wie Software-Design, Implementierung, Tests und Komponentenintegration a priori fest, ihre konkrete Ausprägung zur Laufzeit dagegen nicht. So können Anordnungsbeziehungen zwischen Aufgaben nur zum Teil vorgegeben werden. Um solche Prozesse ohne Einschränkung der Prozessdynamik verwalten und leiten zu können, muss sich einerseits die Prozessstruktur dynamisch an eine sich verändernde Produktstruktur anpassen lassen. Anderseits sind komplexe Kooperationsmechanismen zwischen Software-Entwicklern notwendig, um sobald als möglich vernünftige zwischenzeitliche Ergebnisse liefern zu können.

Die Anforderungen ähneln denen aus Teil I der Arbeit. Im Rahmen zweier **AHEAD**-Anwendungsprojekte SUKITS [HW98b, HW98a] und IMPROVE [JKN+98]) wird außerdem versucht, die **AHEAD**-Konzepte im Maschinenbau (Verfahrens- und Fertigungstechnik) zur Koordination von Produktentwicklungsprozessen einzusetzen.

### 19.1.2 Die AHEAD-Konzepte

Die Grundidee des **AHEAD**-Ansatzes ist die *personenzentrierte* Koordination von Entwicklern auf der Basis einer integrierten Betrachtung von Produkten, Aktivitäten und Ressourcen [Wes01]. Das **AHEAD**-Modell setzt sich deshalb aus drei Hauptkomponenten zusammen:

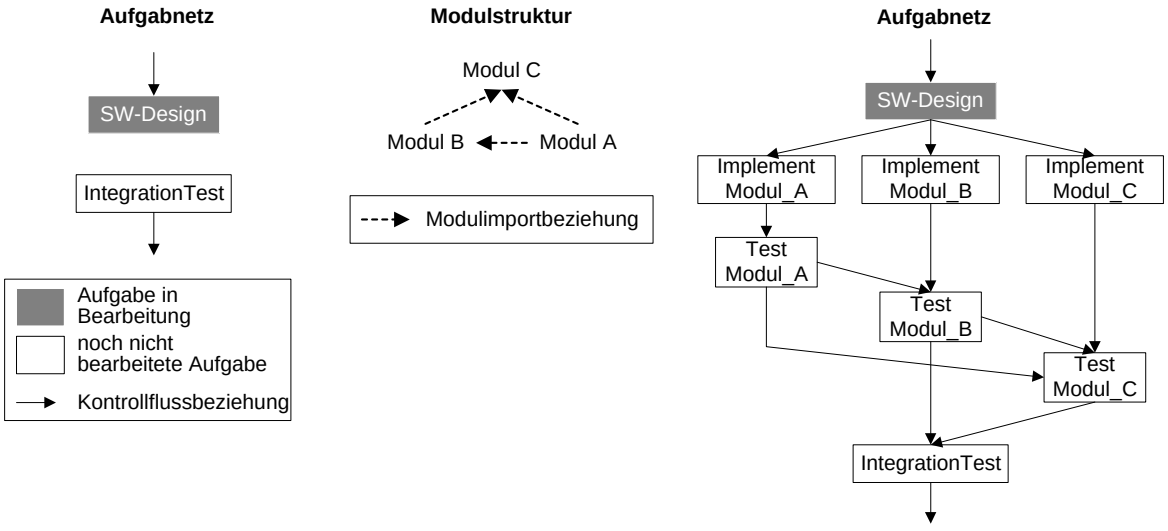
- Konfigurationsmodell **CoMa** (configuration management) [Wes96] zum Konfigurieren und Versionieren von Produkten
- Aufgabenmodell **DYNAMITE** (dynamic task nets) [HJKW95, HJKW96, HJKW97] zum verschränkten Planen und Steuern von Entwicklungsprozessen.
- Ressourcenmodell **RESMOD** (resource model [KKSW98] zum Verwalten von menschlichen und technischen Ressourcen.

Diese Untersuchung konzentriert sich jedoch im Wesentlichen auf **DYNAMITE**-Konzepte, da nur sie für den nachfolgenden Vergleich benötigt werden.

Beim **DYNAMITE**-Ansatz versucht man, die oben genannten Anforderungen mittels *evolutionärer Aufgabennetze* zu verwirklichen. Aufgabennetze basieren auf der Theorie der Graphgrammatiken [SWZ95] beziehungsweise Graphersetzungssysteme [Roz97, Göt98]. Graphgrammatiken ermöglichen die Beschreibung komplexer Graphstrukturen. Mittels Graphersetzungsregeln kann das dynamische Verhalten der Graphen festgelegt werden. Eine Ersetzungsregel beschreibt den Austausch eines Subgraphen mit einem bestimmten Muster durch einen anderen Subgraphen. Ihre Anwendung kann an bestimmte Bedingungen geknüpft werden, wodurch es möglich wird, dass eine Änderung nur in gewissen Zuständen anwendbar ist. Aufgabennetze können als höherwertige Darstellungsform von Graphgrammatiken betrachtet werden, deren Ausführungssemantik durch Graphersetzungsregeln

beschrieben ist. Jede Zustandsänderung einer Aufgabe eines Netzes wird dann auf eine Graphersetzungsregel abgebildet, die zu einem modifizierten Aufgabennetz führt.

Beispielsweise kann ein initiales Aufgabenetz die in Abbildung 19.1 (a) dargestellte Form besitzen. Abhängig vom Ergebnis der Aufgabe *SW-Design* wird das Netz nach den fest vorgegeben Graphersetzungsregeln expandiert. Liefert die SW-Design-Aufgabe die in Abbildung 19.1 (b) gezeigte Modulstruktur aus drei Modulen, so wird das initiale Aufgabennetz zur Laufzeit durch Benutzerinteraktionen des Projektmanagers zu dem in Abbildung 19.1 (c) gezeigten Aufgabennetz expandiert, in dem für jedes Modul eine Implementierungs- und Testaufgabe in das Aufgabennetz eingefügt wird [Wes01, HJKW97].




(a) Initiales Aufgabennetz (b) Aktuelle Modulstruktur (c) Expandiertes Aufgabennetz

**Abbildung 19.1:** Beispiel eines Aufgabennetzes in **DYNAMITE**

Wie aus Abbildung 19.1 ersichtlich können Kontrollflussabhängigkeiten zwischen den Aufgaben eines Aufgabennetzes existieren. Alle möglichen Kontrollflussabhängigkeiten werden neben anderen Aspekten in einem *spezifischen **DYNAMITE**-Modell* festgelegt. Es subsummiert damit alle erlaubten Graphersetzungsregeln und stellt die oben erwähnte höherwertige Darstellungsform dar. Jede zur Laufzeit durchgeführte Veränderung des Aufgabennetzes muss konform zu dem vorgegebenen spezifischen Modell sein. Es entspricht damit in etwa dem Workflow-Schema anderer Workflow-Ansätze, da es alle erlaubten Ausführungsreihenfolgen vorgibt.

Ein spezifisches Modell besteht im Wesentlichen aus Aufgabentypen, zwischen denen Kontroll- und Datenflussabhängigkeiten definiert werden. Für jeden Aufgabentyp ist die minimale und maximale Anzahl Instanzen angegeben, die während der Ausführung des Aufgabennetzes erzeugt werden können. Die Spezifikationen der Eingabe- und Ausgabeparameter bestehen jeweils aus dem Namen, dem Datentyp, der im Konfigurationsmodell festgelegt wird, und der minimalen und maximalen Anzahl aktueller Parameter dieses Typs, die eine Aufgaben-Instanz besitzen kann (*variable Parameterleisten*).

Es sind zwei Arten von Kontrollflüssen möglich: Neben der „normalen“ Kontrollflussbeziehung, die immer in die Ablaufrichtung des Prozesses verweist, enthält das Meta-Modell noch Rückgriffbeziehungen (*Feedback-Kanten*), mit denen sich Rückgriffe und Zyklen im Aufgabennetz beschreiben lassen. Sie sind entsprechend in die entgegengesetzte Richtung orientiert. Datenflüsse werden durch Verknüpfung von Ein- und Ausgabeparameter spezifiziert.

Zur Laufzeit werden den einzelnen Aufgaben eines Aufgabennetzes noch Ressourcen (Personen, Systeme) zugewiesen. Dies erfolgt interaktiv durch den Aufgabennetzmanager, der die schrittweise Ausgestaltung des Aufgabennetzes durchführt.

### 19.1.3 Umsetzung der Prozessanforderungen aus der Produktentwicklung

Obwohl bei der **AHEAD**-Konzeption ähnliche Anforderungen zugrunde gelegt wurden wie die in Teil I beschriebenen Anforderungen zur Unterstützung von Produktentwicklungsprozessen, so stellt man bei genauerer Betrachtung fest, dass im **AHEAD**-Modell die Anforderungen der Produktentwicklungsprozesse nur unzureichend umgesetzt werden können:

*Unterstützung unstrukturierter Teilprozesse innerhalb vorgegebener Prozessstrukturen:*

Die Ausdrucksmächtigkeit des spezifischen Modells von **AHEAD** ist zu beschränkt, um den strukturierten Anteil von Produktentwicklungsprozessen adäquat abzubilden, da nur Sequenzen und Parallelität modelliert werden können. Bedingte Verzweigungen sind gar nicht, Schleifen nur begrenzt mittels Rückgriffbeziehungen möglich, die einzig einen informellen Datenfluss erlauben.

Unstrukturierte Teilprozesse können zwar prinzipiell als Subaufgabennetz ohne Kontrollflussabhängigkeiten realisiert werden. Es entfällt damit aber auch jedwede Benutzerführung, da diese ausschließlich über den Kontroll- und Datenfluss erfolgt. Eine Steuerung über zeitliche Vorgaben wie im **WEP**-Modell ist nicht vorgesehen.

*Prozesskoordiniertes Simultaneous-Engineering:*

Simultaneous-Engineering durch vorzeitige Datenweitergabe ist im **AHEAD**-Ansatz enthalten. Eine Aufgabe kann über ihre Eingabeparameter jederzeit neue Daten empfangen, die als vorläufige Ausgabedaten von einer anderen Aufgabe erzeugt wurden. Es fehlen allerdings Möglichkeiten zur Beschreibung der Vorläufigkeit im Konfigurationsmodell von **AHEAD** (vergleiche Datenqualitätsstufen im **WEP**-Modell) und geeignete Konsistenzsicherungsmechanismen beim Eintreffen aktuellerer Daten. Das **AHEAD**-Modell stellt einzig sicher, dass eine Aufgabe mit veralteten Eingabedaten nicht endgültig beendet werden kann.

Auch sind keine Konzepte zur Unterstützung von Ad-hoc-Benutzerinteraktionen vorhanden, wie sie das **WEP**-Modell in Form von Konsolidierungsrunden bereitstellt.

*Dynamische Anpassung einer vorgegebenen Prozessstruktur:*

Eine Anpassung der Prozessstruktur ist mittels Graphersetzungsregeln möglich. Es muss allerdings bereits zur Modellierungszeit eine fest vorgegebene Obergrenze für die Anzahl paralleler Aufgaben angegeben werden. Diese Einschränkung lässt sich in den meisten Fällen sicherlich durch eine großzügig gewählte Obergrenze umgehen.

Schwerwiegender ist dagegen, dass jede dynamische Anpassung durch den Aufgabennetzmanager interaktiv erfolgen muss. Eine automatische Prozessanpassung ist nicht vorgesehen. Die Graphersetzungsregeln sichern dabei lediglich die Konsistenz des Aufgabennetzes

während der dynamischen Netzumgestaltung. Bei der Festlegung der Graphersetzungsregeln kann im **AHEAD**-Modell nur auf die variable Anzahl der Ausgabeparameter einer Vorgängeraufgabe Bezug genommen werden. Die Angabe komplexerer Regeln, die das Auswerten von Attributwerten, Qualitätsstufen und Subobjektbeziehungen erlauben (vergleiche **WEP**-Traversierungsmerkmale), sind nicht vorhanden. Der graphgrammatik-basierte Ansatz erlaubt es prinzipiell, solche Regeln über Graphersetzungsregeln basierend auf komplex strukturierten Ausgabeobjekten nachzubilden. Die Verwendung dieser Möglichkeit wurde allerdings in den zugänglichen Veröffentlichungen nicht erwähnt.

*Unterstützung adaptiver Projektplanung und Korrektheitsaspekte:*
Die Beachtung zeitlicher Vorgaben aus einem übergeordneten Projektmanagement oder die Zusicherung dynamischer Eigenschaften wie konsistente Datenversorgung oder Terminierungszusicherungen werden in **AHEAD** nicht behandelt.

Zusammenfassend lässt sich damit sagen, dass das **AHEAD**-Modell vielversprechende Ansätze enthält. Bei genauerer Betrachtung kann es Produktentwicklungsprozesse jedoch nur unzureichend unterstützen, da wichtige Aspekte fehlen (Zeitmanagement, Korrektheitseigenschaften) oder nur eingeschränkt verwendet werden können (dynamische Anpassungen, Simultaneous-Engineering-Phasen). Bisher fehlt auch bei auf Graphgrammatik basierenden Modellen der Nachweis ihrer praktischen Einsatzfähigkeit zur Steuerung komplexer Entwicklungsprozesse.

Die Ergebnisse mehrerer anwendungsorientierter Projekte [EPW94, VW00, GS00, KFW01] und die Praxiserfahrungen des Autors aus verschiedenen Fahrzeugentwicklungsprojekten lassen auch an der Grundannahme des **AHEAD**-Ansatzes, dass Entwicklungsprozesse nicht vormodellierbar sind, zweifeln: Ändert sich nicht grundlegend die eingesetzte Technologie, so ist auch jeder Produktentwicklungsprozess in seiner Grobstruktur vormodellierbar und damit im Groben planbar, was für den Unternehmenserfolg auch unabdingbar ist. So haben sich beispielsweise die groben Prozessphasen (Konzept-Design, Programmverifikation, Produkt/Fertigungsprozessplanung und -entwicklung, Produkt/Fertigungsprozessverifikation und -optimierung, Produktionsstart) in der Fahrzeugindustrie trotz Einsatz neuer Entwicklungstechnologien (CAD, CAE, ...) in ihrer prinzipiellen Anordnung in den letzten Jahrzehnten nicht geändert. Lediglich die zeitliche Verschränkung der Prozessphasen hat aufgrund von Simultaneous-Engineering-Anforderungen zugenommen. Dies wird sicherlich auch in anderen produzierenden Industriezweigen gelten, bei denen sich über viele Produktentwicklungen ein etabliertes Vorgehen herauskristallisiert hat.

Eine vollständige Neugestaltung mittels evolutionärer Aufgabennetze macht damit wenig Sinn. Sie ist darüber hinaus insbesondere bei langdauernden Entwicklungsprozessen zu aufwändig. In der Praxis greift man nahezu immer auf vorangegangene Entwicklungsprozesse zurück und modifiziert diese gegebenenfalls anhand der gemachten Erfahrungen („lessons learned“) im Detail. Die aufwändige interaktive Modellierung für jede Prozess*instanz* wird auch als Grund für die Einführung von vordefinierten Standardtypen von Aufgabennetzen angeführt [Wes01].

Die Stärke des **DYNAMITE**-Modells liegt in dessen großer Konfigurierbarkeit [Kra96]. So kann beispielsweise die Schaltlogik für jeden Aufgabentyp einzeln spezifiziert werden. Damit lässt sich prinzipiell das Schaltverhalten klassischer Workflow-Management-Systeme wie auch das Simultaneous-Engineering unterstützende Modell des **WEP**-Ansatzes – wenn auch mit erhöhtem Aufwand – konfigurieren.

Diese Stärken kommen dann zum Tragen, wenn man bei einer Produktentwicklung „völliges Neuland“ betritt, wo sich also noch keine anerkannte Vorgehensweise entwickelt hat. Hier ist dann der erhöhte Modellierungs- und Managementaufwand vertretbar.

## 19.2 Petrinetzbasierte Modelle

### 19.2.1 Einführung und allgemeine Bewertung

Petrinetze [Pet62, Rei85, Rei86, Bau96] wurden ursprünglich als graphischer Formalismus zur Beschreibung und Analyse nebenläufiger Systeme entwickelt. Später wurde begonnen, Petrinetze zur Beschreibung von Arbeitsabläufen (**LEU** [Gru93], **SPADE** [BFGG91] und **INCOME/WF** [OSS+97]) zu verwenden. Im Laufe der Zeit wurden viele verschiedene Petrinetzvarianten entwickelt. Zur Beschreibung von Workflow-Schemata werden jedoch fast ausschließlich (Erweiterungen von) Prädikat-Transitionsnetzen verwendet.

Als wichtigste Argumente für die Verwendung von Petrinetzen werden dabei die fundierte Petrinetztheorie und die Verfügbarkeit von Analyseverfahren genannt [Aal97]. Dem stehen jedoch auch verschiedene Nachteile gegenüber:

Hier ist zum einen die mangelhafte Trennung semantisch unterschiedlicher Modellierungsaspekte, wie beispielsweise Kontroll- und Datenfluss, zu nennen. Aufgrund der wenigen und einfachen Grundelemente von Petrinetzen (Stellen, Transitionen, Marken) müssen diese Aspekte implizit, meist durch verschiedene Markenbezeichnungen, beschrieben werden. Wenige Grundelemente machen zwar die Entwicklung von Verifikationsverfahren einfach, führen aber schnell zu unübersichtlichen und kaum mehr verständlichen Prozessmodellen.

Verifikationsverfahren überprüfen das dynamische Verhalten des durch das Petrinetz beschriebenen Arbeitsablaufs. Alle Verfahren bauen auf dem einfachen Ausführungsmodell auf, bei dem eine Petrinetz-Transition genau dann und genau einmal schaltet, wenn die Bedingungen ihres Vor- und Nachbereichs erfüllt sind. Zur Unterstützung von Simultaneous-Engineering-Phasen werden komplexere Ausführungsmodelle benötigt, die das mehrmalige Schalten derselben Transition erlauben (vergleiche **WEP**-Ausführungsmodell in Kapitel 13). Ein solches Schaltverhalten (einer Transition) kann, wenn überhaupt, höchstens als eigenes Petrinetz nachgebildet werden[1], wodurch die Komplexität der entstehenden Petrinetze bereits bei einfachen Entwicklungsprozessen völlig explodieren würde.

Die geforderte zeitliche Führung innerhalb unstrukturierter Teilprozesse kann mit Petrinetzen ebenfalls nicht dargestellt werden. Es existieren zwar Ansätze, sogenannte *zeitbehaftete Petrinetze*, bei denen Zeitaspekte berücksichtigt werden [RH80, MCB84, KGZH95, LM88, VN88]. Bei diesen Petrinetzvarianten schalten Transitionen zeitverzögert, meist festgelegt durch stochastische Verteilungsalgorithmen. Sie werden bisher ausschließlich zur Performance-Bewertung von (IT-) Systemvarianten verwendet. Auch bei zeitbehafteten Petrinetzen schaltet eine Transition nur einmal. Das im **WEP**-Modell durch Meilensteine ausgelöste mehrmalige „Schalten von zielorientierten Aktivitäten“ ist nicht direkt möglich.

Ein weiteres Grundproblem fast aller Netzvarianten ist das Fehlen von Anpassungsmöglichkeiten eines Petrinetzes zur Laufzeit [Rei00]. Eine für die Anforderungen aus Teil I interessante Ausnahme bildet der **INCOME**-Ansatz, weshalb dieser hier nun detaillierter vorgestellt wird.

[1] Eine Aufgabe eines Entwicklungsprozesses entspricht dann bereits einem eigenen Petrinetz und nicht nur einem Workflow-Schritt. Dagegen kann im **WEP**-Modell eine Aufgabe auf eine zielorientierte Aktivität abgebildet werden.

### 19.2.2 INCOME/STAR und INCOME/WF

Der **INCOME**-Ansatz wurde im Rahmen zweier Forschungsprojekte entwickelt: Zielsetzung des **INCOME/STAR**-Projekts ist die Bereitstellung einer „Rechnergestützten Wartungs- und Entwicklungsumgebung für verteilte betriebliche Informationssysteme“ [SOZ95]. Hierbei wird insbesondere auf eine flexible Unterstützung des Software-Entwicklungsprozesses großer Wert gelegt [Obe94a, Obe94b, Jae96]. Im Rahmen des Folgeprojekts **INCOME/WF** werden die erzielten Ergebnisse aufgenommen und mit dem Ziel erweitert, beliebige semi-strukturierte Prozesse mittels einer Petrinetzbasierten Workflow-Umgebung zu unterstützen [OSS+97].

Zum Erreichen dieser Ziele wurden NR/T-Netze ($NF^2$-Relationen/Transitions-Netze) als Erweiterung von Prädikat-Transitionsnetzen konzipiert [OSS93, Obe94a]. Wesentlicher Vorteil von NR/T-Netzen ist die integrierte Modellierung von Arbeitsabläufen und komplexen $NF^2$-Objektstrukturen [PD89], wodurch Nebenläufigkeit auf komplex strukturierten Objekten (z. B. Büroformulare, Stücklisten) beschrieben werden kann. In NR/T-Netzen besitzen die Marken eine $NF^2$-Struktur zur Repräsentation komplexer Objekte. Nebenläufigkeit wird dadurch erreicht, dass mit Hilfe von *Filtertabellen Teile einer Marke* selektiert und zum Schalten einer Transition verwendet werden können.

Werden beim Auswerten einer Filtertabelle mehrere Teilmarken als „Treffer“ identifiziert, so werden dynamisch mehrere Transitionen erzeugt und jeder Transition eine Teilmarke zugewiesen [Obe94b]. In ähnlicher Form können Teilmarken wieder zu einer Gesamtmarke zusammengefügt werden. Mit diesem Konzept kann ein zum **WEP**-Konstrukt der dynamischen Parallelität vergleichbarer Effekt erzielt werden, um abhängig von der Produktstruktur, bei NR/T-Netzen repräsentiert durch eine strukturierte Marke, die Prozessstruktur anzupassen.

Im Nachfolgeprojekt **INCOME/WF** werden NR/T-Netze zur Modellierung und Ausführung von semi-strukturierten Workflows verwendet. Bei dem dort entwickelten Vorgehensmodell wird der strukturierte Ablauf zuerst grob als *Core Workflows* beschrieben. Während der Ausführung von Core Workflows werden Bearbeitern langdauernde und eventuell unstrukturierte Workflow-Schritte zugewiesen. Sie entscheiden dann individuell, wie sie ihre Aufgaben abarbeiten: Sie können dabei einen für sie *privaten* Workflow modellieren und später wiederverwenden, übliche Groupware-Mechanismen benützen oder verschiedene Regeln (Event-Condition-Action-Rules) formulieren, um über das Auftreten relevanter Ereignisse automatisch informiert zu werden. Die Rückmeldung an den strukturierten Core Workflow erfolgt – im Gegensatz zum **WEP**-Modell – erst mit Beendigung des Workflow-Schritts. Sinnvolle Zwischenergebnisse eines unstrukturierten Workflow-Schritts können deshalb nicht bereits an Folgeschritte weitergegeben werden. Die geforderte Verkürzung von Prozessdurchlaufzeiten durch Unterstützung von Simultaneous-Engineering kann hiermit nicht erzielt werden.

Ansonsten besitzen NR/T-Netze die gleichen bereits im vorherigen Abschnitt beschriebenen Nachteile der Petrinetze, so dass alle weiteren Anforderungen aus Teil I nicht adäquat umgesetzt werden können.

## 19.3 Allgemeine Graphmodelle

Wesentlicher Vorteil petrinetzbasierter Ansätze ist ihr mathematisch fundiertes Modell, welches das Schaltverhalten präzise festlegt und deshalb verifizierbar macht. Verwendet man andere Graphmodelle zur Prozessbeschreibung [KR91, LA94, Jab94, Sie97, RD98], so müssen die formalen Grundlagen

selbst entwickelt werden. Diesem Nachteil steht der Vorteil gegenüber, dass die Modellierungssprachen semantisch reichhaltiger und anwendungsnaher gestaltet werden können, was die Prozessmodellierung vereinfacht und auch die Ausführungssemantik den jeweiligen Anforderungen der Anwendungsdomäne angepasst werden kann.

Diesen Zielkonflikt lösen einige Ansätze, indem sie dem Workflow-Modellierer ein anwendungsnahes Graphmodell zur Beschreibung der Prozesse zur Verfügung stellen, das intern auf Petrinetze abgebildet wird. Verifikation und Ausführung der modellierten Prozesse erfolgt dann auf der internen Petrinetz-Ebene. Die Ausführungsmodelle dieser Ansätze weisen damit die gleichen Schwachstellen wie die bereits diskutierten reinen Petrinetz-Verfahren auf und werden deshalb hier nicht weiter betrachtet. Die folgende Untersuchung konzentriert sich auf Vorschläge, die auch zur Ausführungszeit nicht auf Petrinetze zurückgreifen.

### 19.3.1 Entwurfsorientierte Prozesssteuerung in CONCORD

Bei entwurfsorientierten Ansätzen zur Prozesssteuerung steht die hierarchische Zerlegung einer Entwurfsaufgabe in Subaufgaben im Vordergrund. Diese Aufsplittung orientiert sich im Wesentlichen an der (zukünftigen) Struktur des Entwurfsobjekts. In der Regel kann für jedes Entwurfsobjekt ein Workflow-Schema spezifiziert werden, das dem Entwickler entlang einer Entwurfsmethodik führt und von administrativen Aufgaben wie beispielsweise das Veröffentlichen von Entwurfsergebnissen im Entwicklerteam entlastet. Der fortschrittlichste Ansatz stellt dabei das **CONCORD**-Modell dar, das im nächsten Abschnitt als Repräsentant entwurfsorientierter Konzepte detaillierter diskutiert wird, bevor am Ende dieses Kapitels andere entwurfsorientierte Ansätze kurz beleuchtet werden.

#### 19.3.1.1 Zielsetzung des CONCORD-Modells

Der Anspruch des **CONCORD**-Ansatzes (controlling cooperation in design environments) [MHR96] ist die umfassende und flexible Unterstützung von (Designer-)Teams beim Entwurf komplexer Produkte. Als Anwendungsgebiete werden unter anderem genannt: Komponentenentwurf [RMHN94, RM97] und VLSI-Design [RMH+94].

#### 19.3.1.2 Das CONCORD-Framework

Die Entwurfsunterstützung lässt sich in drei logische Ebenen gliedern, die in Abbildung 19.2 dargestellt sind.

*Administrations- und Kooperationsebene (Administration/Cooperation Level):*

Auf der Administrations- und Kooperationsebene kann eine komplexe Design-Aufgabe interaktiv und schrittweise in für einzelne Entwerfer praktikable Teilaufträge, genannt *Design-Aktivitäten (Design Activities)* zerlegt werden. Neben dieser hierarchischen Zerlegung von Design-Aufgaben kann auch die Weitergabe (vorläufiger) Entwurfsdaten zwischen Bearbeitern verschiedener Design-Aktivitäten mittels *Usage-Kanten* und die Aufforderung zur Abstimmung über *Negotiation-Kanten* spezifiziert werden. Für jede Design-Aufgabe wird ein Entwurfsziel definiert, das aus einer Menge geforderter Eigenschaften, sogenannten *Features* [KRS98], zusammengesetzt ist. Am Ende einer Design-Aktivität muss das ihr zugeordnete Entwurfsobjekt die geforderten Eigenschaften erfüllen.

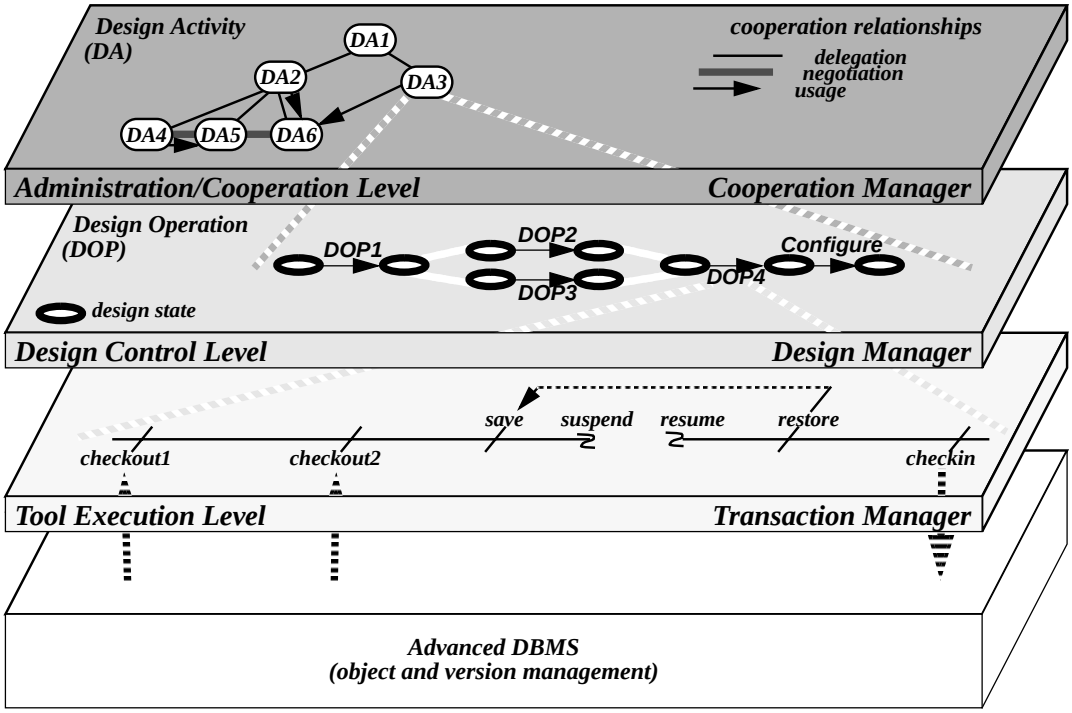



**Abbildung 19.2:** Das **CONCORD**-Framework (aus [RMH96])

*Plan-Ebene (Design Control Level):*

Auf dieser Ebene wird die interne Struktur einer Design-Aktivität spezifiziert. Hier steht die zeitliche Anordnung atomarer Entwurfsschritte in einem Workflow zum Erreichen eines Entwurfsziels im Vordergrund. Ziel dieser in **CONCORD** mit *Designflows* bezeichneten Workflows ist die Entlastung der Entwickler von Routinetätigkeiten. Als Modellierungsprimitive von Designflows stehen Sequenzen, Schleifen, bedingte und parallele Verzweigungen sowie als Workflow-Schritte die *Design-Operationen (Design Operations, DOP)* zur Verfügung. Eine Design-Operation besteht aus einer Vor- und Nachbedingung sowie einer Aktion, die entweder ein einfacher Werkzeugaufruf (z. B. CAD-System) oder eine abstrakte Aktivität repräsentieren. Bei einer abstrakten Aktivität bleibt es dem Bearbeiter selbst überlassen, wie er die Nachbedingungen seiner Design-Operation erreicht.

*Werkzeug-Ebene (Tool Execution Level):*

Diese Ebene steuert einzelne Werkzeugaufrufe. Sie kapselt die realen Werkzeuge durch Wrapper. Aus Sicht der Datenhaltung wird eine Werkzeuganwendung als Transaktion (mit recovery points) betrachtet, die Objektversionen erzeugt, liest und verändert [Rit96, KRS98].

#### 19.3.1.3 Umsetzung der Prozessanforderungen aus der Produktentwicklung

Im Folgenden wird untersucht, wie sich die in Teil I identifizierten Anforderungen bezüglich einer Prozessunterstützung für die Produktentwicklung mit den **CONCORD**-Konzepten umsetzen lassen.

*Unterstützung unstrukturierter Teilprozesse innerhalb vorgegebener Prozessstrukturen:*

Schwerpunkt des **CONCORD**-Modells bildet die *vertikale* Kooperation „überschaubarer“ Entwicklerteams durch Dekomposition von Aufgaben in Subaufgaben. Für die horizonta-

le Prozesskoordination der Entwicklungspartner entsprechend vorgegebener Prozessstrukturen ist der **CONCORD**-Ansatz dagegen weniger geeignet. Notwendig hierfür ist eine flexible Workflow-Unterstützung über verschiedene Design-Aktivitäten. Eine solche Workflow–Koordination kann zwar durch Negotiation-Kanten zwischen Design-Aktivitäten spezifiziert werden und damit in der Plan-Ebene ein *kooperativer* Designflow [RM97] angefordert werden. Die Festlegung von Negotiation-Kanten kann jedoch nicht vormodelliert, sondern nur interaktiv zur Laufzeit erfolgen.[2] Damit sind die möglichen Negotiation-Verknüpfungen von Design-Aktivitäten zur Modellierungszeit von Designflows natürlich nicht bekannt. Die Modellierung kooperativer Designflows kann sich deshalb nur auf einfache und generisch einsetzbare (Verhandlungs-)Workflows beschränken. Komplexere Entwicklungsprozesse, die mehr als ein paar Design-Aktivitäten umfassen, sind aufgrund ihrer kombinatorischen Vielfalt durch diesen Ansatz nicht möglich.

Designflows können als Prozessschritte abstrakte Aktivitäten beinhalten, die zur Repräsentation eines unstrukturierten Teilprozesses verwendet werden können. Da jedoch Möglichkeiten zur zeitlichen Steuerung fehlen, kann das **CONCORD**-System den Benutzer nicht führen (vergleiche das Meilenstein-Konzept des **WEP**-Modells). Auch fehlen Konzepte zur Konsistenzsicherung auf der Planebene.

*Prozesskoordiniertes Simultaneous-Engineering:*
Auf der Administrations- und Kooperationsebene wird durch das interaktive Verbinden zweier Design-Aktivitäten durch Usage-Kanten die Möglichkeiten für den Austausch vorläufiger Daten zwischen den verbunden Design-Aktivitäten geschaffen. Zum **WEP**-Ansatz vergleichbare Benutzeroperationen zum Weitergeben (*Permit*), Zurückziehen (*Revoke*) und Anfordern (*Request*) von Daten sind vorhanden [RM97]. Teilweise sind weitergehende Laufzeitoperationen, wie zum Beispiel die Übertragung des Besitzrechts von Objektversionen, möglich. Der im Vergleich zum **WEP**-Modell höheren Flexibilität steht jedoch die fehlende Ausführungssicherheit gegenüber. So werden beispielsweise zyklische Datenweitergaben nicht überwacht.

*Dynamische Anpassungen vorgegebener Prozessstrukturen, Unterstützung adaptiver Projektplanung:*
Weder eine dynamische Prozessstrukturanpassung von Designflows noch die Berücksichtigung zeitlicher Planungsvorgaben sind im **CONCORD**-Modell vorgesehen.

*Sicherstellung einer korrekten Prozesssteuerung.:*
Das Fehlen einer design-aktivitäts-übergreifenden Workflow-Steuerung macht die Zusicherung von Korrektheitsaspekten auf der Administrations- und Kooperationsebene nicht möglich. Korrektheitsgarantien werden lediglich auf der Werkzeug-Ebene (transaktionsgeschützte Manipulation von Objektversionen und -konfigurationen) [Rit96, RMH$^+$94] zugesichert. Eine Äquivalenz zur garantierten Workflow-Terminierung mit konsistenten Prozess und Nutzdaten (vergleiche SE-Serialisierbarkeit im **WEP**-Modell) fehlt dagegen.

Beim detaillierten Vergleich der **CONCORD**-Ansätze mit den **WEP**-Konzepten werden somit die unterschiedlichen Schwerpunkte deutlich: Während die Hauptzielrichtung des **WEP**-Modells die flexible horizontale Prozess*koordination* aller am *Entwicklungs*prozess beteiligten Personen ist, konzentriert sich das **CONCORD**-Projekt auf *Kooperationsmechanismen* beim *Konstruktions*prozess [Ehr95, Pah99] mit dem Ziel, Entwicklungsteams in ihrer Konstruktionsmethodik zu unterstützen. Unter diesem Gesichtspunkt kann der **CONCORD**-Ansatz eine Methodik zum Erreichen der Ziele bestimmter **WEP**-Aktivitäten, wie beispielsweise der *DefineChangeRequest*-Aktivität im **WEP**-Workflow aus Abbildung 8.7 (vergleiche auch Abbildung 2.2) bereitstellen.

[2] Eine Vormodellierung würde auch nicht der Philosophie des **CONCORD**-Ansatzes entsprechen.

#### 19.3.1.4 Weitere entwurfsorientierte Ansätze

Zu **CONCORD** ähnliche, wenn auch nicht so weitreichende, Ansätze stellen **CRISTAL** [BBC+98] und **SIFRAME** [Kes96] dar. **SIFRAME** wurde zum kommerziellen Produkt weiterentwickelt, konnte sich allerdings nicht am Engineering -Markt durchsetzen.

Beim **CRISTAL**-Modell wird auf die Administrations- und Kooperationsebene des **CONCORD**-Frameworks verzichtet. Somit fehlen logischerweise auch die in **CONCORD** möglichen Kooperationsmechanismen. Workflows werden nicht an Aufgaben, sondern direkt beim assoziierten Entwurfsobjekttyp spezifiziert. Jede neue Instanz des Objekttyps stößt diesen Workflow an. Entwurfsobjekttyp-übergreifende Workflows sind nicht möglich.

**SIFRAME** besitzt ebenfalls eine Administrations- und Kooperationsebene, auf der Projekte und Subprojekte mit Bearbeitern und Objektinstanzen verknüpft werden. Weitere Beziehungen, wie die Usage- und Negotiation-Kanten des **CONCORD**-Modells fehlen jedoch.

### 19.3.2 Flexible Workflow-Koordination in ADEPT

Für viele Anwendungen ist die von heutigen kommerziellen Systemen gebotene Workflow-Steuerung zu starr, weil keine Mechanismen vorgesehen sind, die es privilegierten Bearbeitern ermöglichen, von vordefiniertem Workflow abzuweichen [BD96a, BDS98, KFW01, DKR+95, Mey96, RD00]. Inzwischen existieren einige wissenschaftliche Vorschläge, welche die für viele Anwendungen geforderte flexiblere Workflow-Steuerung ermöglichen (z. B. [Sie97, CLK99, CFM99, Rei00]). Der weitestgehende, da konsistenzerhaltende Ansatz stellt sicherlich das **ADEPT**-Modell [Rei00] dar, das im Folgenden als Repräsentant für flexible Workflow-Steuerung detaillierter beleuchtet und danach auf seine Eignung für Produktentwicklungsprozesse analysiert wird.

#### 19.3.2.1 Zielsetzung des ADEPT-Ansatzes

Zielsetzung des **ADEPT**-Ansatzes (Application Development Based on Pre-Modeled Process Templates) [DKR+95, RBFD01] ist es, eine minimale und trotzdem vollständige Menge von Benutzeroperationen bereitzustellen, die Ad-hoc-Modifikationen von Workflow-Instanzen ermöglichen, *ohne* die Konsistenz und Korrektheit des Workflows zu gefährden. Die Sicherung von Konsistenz und Korrektheit ist insbesondere für unternehmenskritische Prozesse von allergrößter Bedeutung. Hier unterscheidet sich das **ADEPT**-Modell positiv von den meisten anderen Ansätzen zur flexiblen Workflow-Ausführung. **ADEPT** erlaubt unter anderem strukturverändernde Modifikationen einer Workflow-Instanz. Darunter sind das Einfügen, Löschen, Verschieben oder Auslassen von Workflow-Schritten, das Serialisieren paralleler Schritte sowie das dynamisch erzeugte Iterieren beziehungsweise Zurücksetzen zu verstehen.

#### 19.3.2.2 Das ADEPT-Basismodell

Alle dieser in **ADEPT** möglichen ad hoc initiierten Modifikationen basieren auf einem formalen Graphmodell mit symmetrischer Blockstruktur, dem ***ADEPT**-Basismodell*, das bereits über sehr umfangreiche Modellierungskonstrukte verfügt [Rei00]. Prozessschritte, im **ADEPT**-Modell als *Knoten*

bezeichnet, werden über Kontrollflusskanten miteinander zu einem vollständigen gerichteten Graphen verknüpft. Neben den üblichen Kontrollflusskonstrukten (Sequenz, bedingte und parallele Verzweigungen, Schleifen) existieren solche zur Beschreibung paralleler Zweige mit finaler Auswahl, Synchronisationskanten zur Festlegung zeitlicher Abhängigkeiten zwischen Knoten aus verschiedenen parallelen Zweigen und Fehlerkanten zum partiellen Zurücksetzen von Workflow-Instanzen mit zu Kompensationssphären [Ley95] vergleichbarer Mächtigkeit. Datenflüsse werden über globale Workflow-Variablen, genannt *Datenelemente*, spezifiziert, die, um partielles Zurücksetzen zu ermöglichen, Datenversionen verwalten. **ADEPT** stellt aufgrund des blockstrukturierten Graphmodells beziehungsweise wegen verschiedener entwickelter Überprüfungsalgorithmen eine datenkonsistente und korrekte Workflow-Ausführung sicher. Die Abbildung 19.3 zeigt einen **ADEPT**-Workflow.

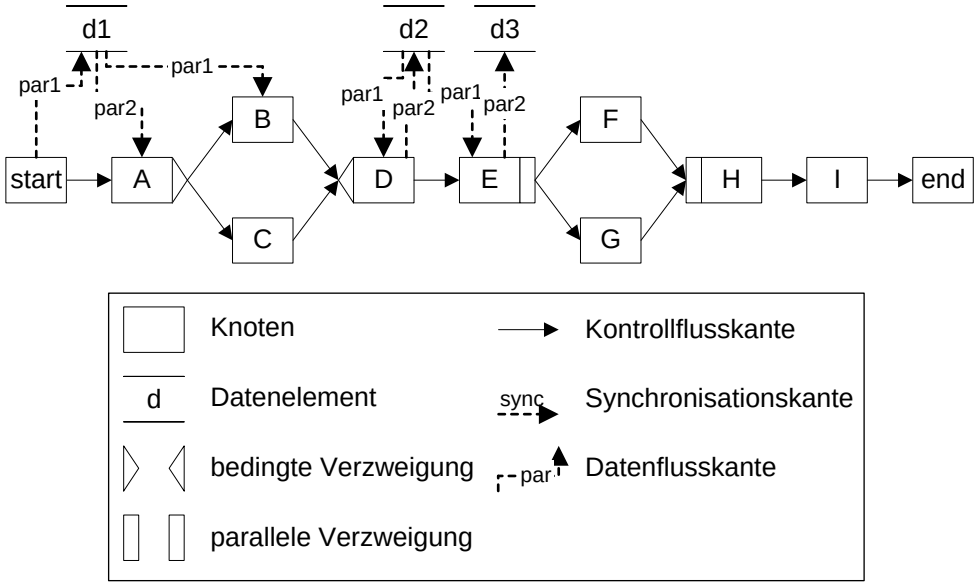



**Abbildung 19.3:** Beispiel eines **ADEPT**-Workflows

Das Schaltverhalten einer **ADEPT**-Workflow-Instanz entspricht prinzipiell dem üblicher Workflow-Ausführungsmodelle. Initialer Bearbeitungszustand einer **ADEPT**-Knoteninstanz ist $NOT\_ACTIVATED$. Sind alle Vorbedingungen erfüllt, so wechselt sie in den Zustand $ACTIVATED$. Während ihrer Bearbeitung befindet sie sich im Zustand $RUNNING$, von dem sie nach erfolgreicher Bearbeitung in den Zustand $COMPLETED$ beziehungsweise bei Abbruch in den Bearbeitungszustand $FAILED$ wechselt. Kann eine Knoteninstanz nicht mehr ausgeführt werden, so terminiert sie mit dem Zustand $SKIPPED$.

#### 19.3.2.3 ADEPT$_{flex}$: Ad-hoc-Modifikationen von Workflow-Instanzen

Eine der wichtigsten Herausforderungen des **ADEPT**-Ansatzes war die Entwicklung von allgemein anwendbaren Benutzeroperationen, um Ad-hoc-Modifikationen an Workflows zur Laufzeit durchführen zu können, ohne deren Konsistenz und Korrektheit zu gefährden [RD97, RD98]. So muss unter anderem bei jeder Veränderung vorab überprüft werden, ob die Datenversorgung noch nicht ausgeführter Workflow-Schritte gewährleistet bleibt und ob die Modifikationen beim gegenwärtigen Bearbeitungszustand des Workflows sinnvoll sind. Letzteres ist beispielsweise nicht der Fall, wenn Knoten in Zweige eingefügt werden oder daraus gelöscht werden, die nicht mehr durchlaufen werden können.

Die entwickelten Algorithmen sind so generisch, dass sie auf beliebige Ausprägungen von **ADEPT**-Workflows angewandt werden können [RD98]. Dabei wird ein mehrstufiges Verfahren benützt, das nun exemplarisch an einer Einfügeoperation gezeigt wird: Es soll in dem **ADEPT**-Workflow aus Abbildung 19.3 der Knoten *X* nach dem Knoten *F* und *G* und vor dem Knoten *I* eingefügt werden. Dazu wird der diese Knoten umfassende minimale Kontrollflussblock bestimmt (Abbildung 19.4 (a)), *X* als zu diesem Block paralleler Knoten eingefügt und anschließend über Synchronisationskanten die geforderte Ausführungsreihenfolge spezifiziert (Abbildung 19.4 (b)). Anschließend kann der Workflow-Graph durch Anwendung verschiedener Reduktionsregeln [Rei00] noch vereinfacht werden (Abbildung 19.4 (c)).

Natürlich können Einfügeoperationen wie auch alle anderen Modifikationen auf bestimmte Bearbeitergruppen, auf spezifische Workflow-Typen oder -Kategorien, auf ausgewählte Bereiche oder Bearbeitungszustände eines Workflows beschränkt werden.

#### 19.3.2.4 Umsetzung der Prozessanforderungen aus der Produktentwicklung

Es wird nun untersucht, wie die **ADEPT**-Konzepte zur Unterstützung von Produktentwicklungsprozessen verwendet werden können.

*Unterstützung unstrukturierter Teilprozesse innerhalb vorgegebener Prozessstrukturen:*

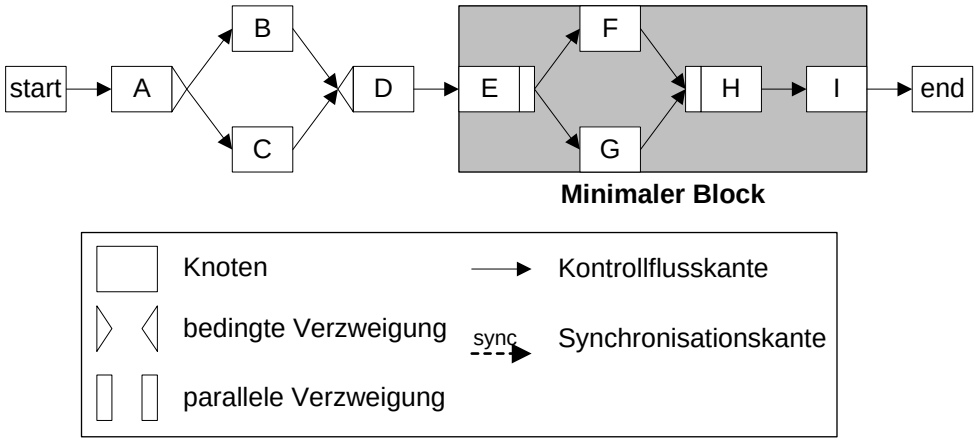
Die in Teil I geforderte Unterstützung unstrukturierter Teilprozesse ist in **ADEPT** nicht vorgesehen. Mittels dynamischer Einfügeoperationen und sogenannter Stoppknoten [RD98] lassen sich Teile der Forderungen jedoch umsetzen. Ein Stoppschritt kann dazu verwendet werden, einen unstrukturierten Teilprozess zu repräsentieren. Dazu muss an diesem Stoppknoten spezifiziert werden, dass das Einfügen auf die Knotentypen beschränkt ist, welche innerhalb des unstrukturierten Teilprozesses ausgeführt werden dürfen und dass dieses Einfügen unmittelbar nach dem Stoppknoten erfolgen muss. Eine geeignete Benutzeroberfläche vorausgesetzt kann damit eine beliebige Ausführungsreihenfolge ad hoc durch den Bearbeiter des Stoppschritts definiert werden.

Es fehlen jedoch Möglichkeiten, die Modifikationen des Datenflusses auf die Eingabe- und Ausgabeparameter des Stoppknotens zu beschränken sowie eine zeitliche und zielgerichtete Führung innerhalb eines unstrukturierten Teilprozesses. Zeitangaben in **ADEPT** beschränken sich auf Vorgaben, wann ein neu eingefügter Prozessschritt bearbeitet werden soll und nicht auf die abzuliefernden Ausgabedaten. Außerdem können bei Ad-hoc-Modifikationen Zeitangaben nicht vormodelliert werden, sondern müssen vorgehensbedingt zur Laufzeit spezifiziert werden, wodurch zumindest ein erhöhter administrativer Aufwand für die Bearbeiter zur Laufzeit verursacht wird.

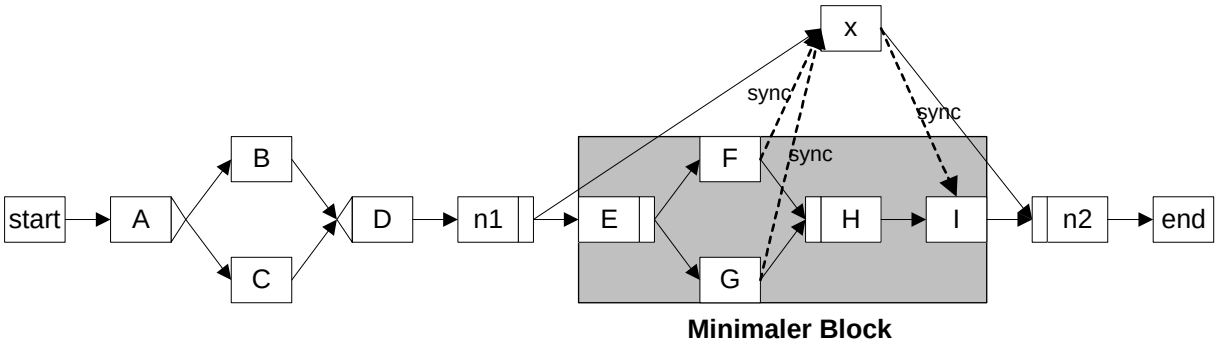
*Dynamische Anpassungen vorgegebener Prozessstrukturen:*

Für die in Teil I geforderte automatische Anpassung der Prozessstruktur anhand der aktuell vorliegenden Objektstruktur werden im **ADEPT**-Modell direkt keine Mechanismen zur Verfügung gestellt. Die für die Analyse komplex strukturierter Daten benötigte Funktionalität kann als zu kodierende Anwendungsprogramme[3] in **ADEPT**-Workflows integriert werden, in dem sie Workflow-Knoten zugewiesen werden, die von (automatischen) Agenten bearbeitet werden. Diese Agenten können dann entsprechend der Analysetreffer des Anwendungs-

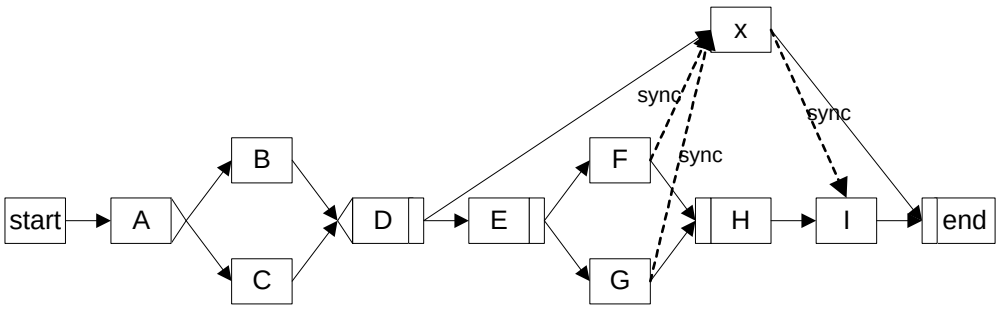
[3] Geeignete generische Anwendungsprogramme könnten als Programmbaustein-Bibliotheken [DKR+95] jeder **ADEPT**-Installation zur Verfügung gestellt werden.




(a) Bestimme den minimalen Block, der *F*,*G* und *I* umschließt.




(b) 1. Füge die Nullknoten *n1* und *n2* als Hilfsknoten vor beziehungsweise nach dem minimalen Block ein.
2. Füge *X* als neuen parallelen Zweig zum minimalen Block ein.
3. Synchronisiere *X* mit *F*,*G* und *I* mittels Synchronisationskanten.




(c) Wende Reduktionsregeln [Rei00] an.

**Abbildung 19.4:** Beispiel einer Einfügeoperation im **ADEPT**-Modell: Füge Knoten *X* nach dem Knoten *F* und *G* und vor dem Knoten *I* in den **ADEPT**-Workflow aus Abbildung 19.3 ein.

programms (vergleiche **WEP**-Traversierungstreffer) über geeignete Einfügeoperationen die Prozessanpassungen inklusive Datenelementzuweisungen automatisch durchführen.

*Prozesskoordiniertes Simultaneous-Engineering:*
Wie bereits in Kapitel 11 aufgezeigt, ist für die Unterstützung von Simultaneous-Engineering ein weitaus komplexeres Ausführungsmodell notwendig als es von herkömmlichen Workflow-Management-Modellen angeboten wird. Die Schaltlogik von **ADEPT** folgt jedoch im Wesentlichen genau diesen einfachen Ausführungsmodellen, bei denen jeder Workflow-Schritt genau dann angeboten und bearbeitet wird, wenn Vorgängerschritte vollständig beendet oder nicht mehr aktivierbar sind und alle notwendigen Eingabeparameter in endgültiger Form vorliegen. Die Weitergabe vorläufiger Informationen vor Beendigung eines in Bearbeitung befindlichen Schritts beziehungsweise das Aktivieren eines Bearbeitungsschritts auf Basis vorläufiger Daten ist nicht vorgesehen und kann in allgemeiner Form auch nicht nachgebildet werden.

Diese fundamentale Voraussetzung für die Unterstützung von Produktentwicklungsprozessen führt somit dazu, dass Anforderungen wie Unterstützung unstrukturierter Teilprozesse und dynamische Anpassungen vorgegebener Prozessstrukturen in praktischen Anwendungen nur begrenzt eingesetzt werden können, obwohl sie – wie bereits aufgezeigt – prinzipiell umgesetzt werden können. Sowohl unstrukturierte Teilprozesse als auch dynamische Prozessanpassungen müssen bereits mit vorläufigen Daten versorgt werden und damit in der Regel mehrmals aktivierbar sein. Dieses Schaltverhalten kann das einfache **ADEPT**-Ausführungsmodell nicht bieten. Es fehlen deshalb auch die für Simultaneous-Engineering notwendigen Konsistenzsicherungsmechanismen.

*Unterstützung adaptiver Projektplanung:*
Im **ADEPT**-Projekt wurde das Zeitmanagement flexibler Workflows nur am Rande im Rahmen einer Diplomarbeit [Gri97] betrachtet. Die dort entwickelten Planungsalgorithmen basieren auf Netzplantechniken, wie sie auch beim **WEP**-Ansatz [Kos00] verwendet werden. Die Algorithmen berücksichtigen sowohl bedingte Verzweigungen als auch Schleifen, sodass sie den Anforderungen aus Teil I im Wesentlichen gerecht werden. Es fehlen allerdings Möglichkeiten, eine zeitliche Führung für unstrukturierte Teilprozesse zu spezifizieren (vergleiche Meilenstein-Konzept des **WEP**-Modells, Abschnitt 6.2).

*Sicherstellung einer korrekten Prozesssteuerung.:*
Eine der herausragenden Leistungen des **ADEPT**-Modells stellt sicherlich die Sicherung von Konsistenz und Korrektheit bei gleichzeitig mächtigen Ad-hoc-Modifikationsmöglichkeiten dar. Die Zusicherung einer korrekten Workflow-Ausführung hängt jedoch immer davon ab, welcher Korrektheitsbegriff zugrunde liegt [GHS95, Ley97]. Formal betrachtet sichert das **ADEPT**-Ausführungsmodell den hier für Produktentwicklungsprozesse geforderten Korrektheitsbegriff der SE-Serialisierbarkeit zu (vergleiche Definition in Kapitel 14), da **ADEPT** kein Simultaneous-Engineering ermöglicht: Ein seriell ausgeführter Workflow wurde natürlich auch SE-serialisiert durchgeführt. Das aus Anwendersicht angestrebte Ziel der Verkürzung der Prozessdurchlaufzeiten durch frühzeitigeren Informationsaustausch und höheren Parallelitätsgrad wird durch solche serielle Ausführungen jedoch nicht erreicht.

Zusammenfassend lässt sich sagen, dass das **ADEPT**-Modell einen sehr interessanten und praxisrelevanten Ansatz darstellt. Ein Einsatz für das mit dem **WEP**-Modell anvisierte Gebiet der Produktentwicklung scheitert jedoch am einfachen **ADEPT**-Ausführungsmodell. Die Kombination der in den jeweiligen Arbeiten entwickelten Konzepte würde sicherlich sehr vielversprechende An-

wendungsmöglichkeiten ergeben. Es sind allerdings noch viele wissenschaftlich sehr anspruchsvolle Herausforderungen zu lösen. Man denke beispielsweise an die Erweiterung des **ADEPT**-Ausführungsmodells um Simultaneous-Engineering-Aspekte oder an die Integration der **ADEPT**-Ad-hoc-Modifikationsmöglichkeiten in den **WEP-**Ansatz.

# Kapitel 20

# Vergleich mit Ansätzen aus dem Management komplex strukturierter Daten

Die technischen Fragestellungen, die sich aus dem Management komplex strukturierter Daten ergeben, umfassen ein sehr breites Spektrum. Darunter fallen Versions- und Konfigurationsmanagement (z. B. [Kat90, KRS98]), Anfrageoptimierung (z. B. [MDEF01]) und Replikationsverfahren für einen effizienten Zugriff auf komplexe und umfangreiche Daten (z. B. [BD96b]) sowie Konzepte zur Integration heterogener Systeme (z. B. [Sau98]), um nur einige zu nennen. Die im Folgenden diskutierten Ansätze enthalten in der einen oder anderen Form Ideen zur Prozesssteuerung oder stellen mögliche Alternativen zu den **WEP**-Konzepten dar. Dem Leser sollte jedoch bewusst sein, dass der Schwerpunkt dieser Arbeiten nicht das Prozessmanagement ist. Eine ähnlich detaillierte Diskussion anhand der Anforderungen aus Teil I ist somit sicherlich weder sinnvoll noch wird sie den Ansätzen gerecht. Da für eine adäquate Gesamtlösung ein integratives Zusammenspiel zwischen Prozess- und Datenmanagement eine wichtige Anforderung darstellt (siehe Teil I), ist es eine wissenschaftlich interessante Aufgabe, die **WEP**-Konzepte mit Lösungsvorschlägen für das Verwalten komplexer Daten zu vergleichen.

## 20.1 Datenqualitäten

Eine wesentliche Voraussetzung von Simultaneous-Engineering ist der Austausch vorläufiger Daten, deren Grad der Vorläufigkeit eindeutig definiert werden muss, damit zwischen Bereitsteller und Empfänger der Daten keine Missverständnisse entstehen. Dies ist der wesentliche Grund, weshalb im **WEP**-Modell das Konzept der Datenqualitätsstufen zur formalen Beschreibung von Datenqualitäten eingeführt wurde.

Der Begriff der Datenqualität wurde in der Literatur bereits häufig und teilweise unter sehr unterschiedlichen Gesichtspunkten diskutiert [WSF95, WW96, Orr98]. Unter anderem auch durch den E-Business-Boom [Hoh99, Dud00, Amo01] hat die Spezifikation von Datenqualitäten neu an Bedeutung gewonnen. Charakteristisch für alle E-Business-Anwendungen ist der unternehmensübergreifende Austausch von Informationen [CS00]. Die daraus resultierende Heterogenität der eingesetzten Sys-

teme und Datenmodelle auf der technischen Seite verknüpft mit der Schnelllebigkeit und Anonymität von E-Business-Anwendungen erfordern eine klare und eindeutige Definition der auszutauschenden Informationen nicht nur bezüglich der Inhaltsstruktur, sondern insbesondere auch bezüglich der Informationsgüte [BS01]. Das schnelle und zuverlässige Bereitstellen von Informationen hoher Qualität ist ein wichtiges Kriterium bei der Auswahl von Geschäftspartnern, da diese Faktoren die internen Prozesse und deren Kosten wesentlich beeinflussen [CS00].

Die in der Literatur zu findenden Ansätze zur Datenqualität beschreiben diese durch die Festlegung von zueinander orthogonalen Kriterien, wie beispielsweise *Exaktheit*, *Vollständigkeit* oder *Konsistenz*. Benutzt werden diese Datenqualitätskriterien im Wesentlichen zur statistischen Analyse großer Datenmengen, beispielsweise bei Datawarehouse-Anwendungen oder bei der nachträglichen Überprüfung ausgetauschter Daten. Es stehen Fragestellungen wie „Wie viele Prozent der (ausgetauschten) Daten waren vollständig?“ im Vordergrund.

Die Verwendung von Datenqualitäten zur Beschreibung und Steuerung von Prozessen, wie im **WEP**-Modell, wurde bisher in der Literatur nicht diskutiert. Ein erster Schritt in diese Richtung stellt der in [BS01] beschriebene Ansatz dar. Dort werden die Kriterien zur Beschreibung von Datenqualitäten in *immanente* und *prozessspezifische* Dimensionen unterteilt. Während erstere den bisherigen Kriterien anderer Ansätze entsprechen, beschreiben letztere die geforderte Datenqualität bei einem spezifischen Datenaustausch zwischen zwei Prozessschritten. Die prozessspezifischen Dimensionen umfassen *Pünktlichkeit*, *Wichtigkeit für den Datenempfänger* und *Zuverlässigkeit des Senders*. Ziel der Einführung dieser prozessspezifischen Qualitätsdimensionen sind wiederum Analysen, um beispielsweise die Pünktlichkeit des Datenbereitstellers für einen bestimmten Prozessschritt beurteilen zu können. Im Gegensatz zum **WEP**-Modell, bei dem die Datenqualitäten *einer* Objektversion überprüft und gegebenenfalls die Weitergabe abgelehnt wird, beziehen sich die Analysen bei diesem Ansatz auf eine a posteriori Beurteilung einer *großen Anzahl* von Prozessinstanzen.

Der einfache ein-dimensionale **WEP**-Ansatz zur Datenqualitätsbeschreibung ist sicherlich nicht so weitreichend wie die mehr-dimensionalen Vorschläge in der Literatur. Für das Verständnis der weiteren **WEP**-Konzepte ist er vollkommen ausreichend. Alle anderen **WEP**-Konzepte sind jedoch auch problemlos mit einem mehr-dimensionalen Datenqualitätsbegriff kombinierbar. Erfahrungen aus der Praxis zeigen jedoch, dass mehr als zwei Dimensionen für den Anwender nur schwer verständlich sind. Derzeit wird bei DaimlerChrysler über die Einführung eines zwei-dimensionalen Datenqualitätsbegriffs für die Motoren- und Getriebeentwicklung diskutiert: Die erste Dimension ist dabei objektklassenspezifisch und entspricht bei der Definition im Wesentlichen dem rekursiven **WEP**-Konzept. Die zweite Dimension beschreibt die *Endgültigkeit* der erreichten Qualitätsstufe der ersten Dimension mit den zwei Werten *vorläufig/endgültig*.

## 20.2 Ergänzung des Datenmanagements um Prozessaspekte

Dieser Abschnitt behandelt Datenmanagementansätze, die um *aktive* Elemente ergänzt wurden, um von einer passiven Informationskomponente zu einem von sich aus agierenden System zu werden, das den Benutzer auf Veränderungen hinweist (vergleiche aktive Datenbanksysteme). Diese aktive Benutzerunterstützung kann vom einfachen Bereitstellen und Synchronisieren konkurrierender Applikationen [Bre95] über Trigger-Mechanismen (**IMLE**, Abschnitt 20.2.1) bis zu datenzentrierten Workflow-Konzepten (Ontologische Netzwerke, Abschnitt 20.2.2, **SIMNET**, Abschnitt 20.2.3) reichen.

### 20.2.1 IMLE

Die stetig steigende Komplexität von Produkten, die stärkere Arbeitsteilung entlang des Entwicklungsprozesses mit der immer weitergehenden Verschränkung unterschiedlicher Technologien (Maschinenbau, Elektrotechnik, Informationstechnologie) zur Realisierung der geforderten Funktionalität führt dazu, dass kein allumfassendes Datenmodell zur Produktbeschreibung existiert. Stattdessen wird jeder Aspekt eines Produkts in einem eigenen Datenmodell und häufig auch in einem separaten System beschrieben. Natürlich bestehen vielfältige Abhängigkeiten zwischen den verschiedenen Elementen der unterschiedlichen Datenmodelle. Die Erhaltung der Datenkonsistenz der Produktinstanzen bereitet in der Praxis häufig große Probleme.

Hier setzt das **IMLE**-Konzept (Intelligent Mechatronic Solution Element) [GCE00] an. Bei diesem Ansatz wird ein Produkt ganzheitlich durch die *hierarchische* Komposition *intelligenter Lösungselemente* beschrieben. Ein Lösungselement beschreibt nur ein Teilprodukt (z. B. Kolben, Zylinderkopf, Zylinderblock), dieses allerdings ganzheitlich mit allen seinen verschiedenen Datenmodellen, den sogenannten *Aspekten* (z. B. geometrischer, Verhaltens-, topologischer Aspekt). Ein Lösungselement ist *intelligent*, wenn Abhängigkeiten zwischen den verschiedenen Datenmodellen explizit modelliert sind, die bei Veränderung eines Aspektinhalts (z. B. neues Material) *Aktionen* bei den jeweiligen anderen betroffen Produktaspekten auslöst. Eine Aktion kann die Benachrichtigung einer Person oder das Anstoßen automatischer Berechnungs- oder Transformationsroutinen sein, um die Datenkonsistenz bei allen Aspekten wiederherzustellen. Durch die hierarchische Strukturierung der Lösungselemente kann eine Detailänderung auf diese Weise bis zum Gesamtprodukt automatisch propagiert werden.

Dieser einfache Trigger-Mechanismus ist sicherlich nicht geeignet, um die aus Teil I geforderte komplexe Prozesssteuerung zu realisieren. Der Ansatz belegt jedoch wiederum die Notwendigkeit einer engen Kopplung zwischen Daten- und Prozessmanagement. Die Veränderung eines Aspekts kann die Anpassung anderer Aspekte bedingen, was wiederum eine dynamische Anpassung des Entwicklungsprozesses notwendig macht. Dynamische Entwicklungsprozessanpassungen werden im **WEP**-Modell unter anderem mit dem Traversierungskonzept ermöglicht. Im Gegensatz zum **WEP**-Ansatz werden beim **IMLE**-Konzept bei der Analyse von Datenabhängigkeiten jedoch keine Datenqualitätsstufen verwendet.

### 20.2.2 Ontologische Netzwerke

Ein aus Sicht der Informationsstrukturierung zum vorherigen **IMLE**-Modell ähnlicher Ansatz wird in [LSK98, WLK00, MLK01] beschrieben. Auch hier wird ein Produkt durch unterschiedliche, aber gleichberechtigte Teilaspekte, genannt *Domains*, beschrieben. Alle Domains zusammen repräsentieren das *Informationssystem* eines Produkts.

Bei der Prozesssteuerung unterscheiden sie sich jedoch: Während sich die Prozesskoordination im **IMLE**-Modell auf einen Trigger-Mechanismus beschränkt, werden bei diesem Ansatz Prozessschritte, sogenannte *Tasks*, aufgrund ihrer benötigten Eingabe- und bereitgestellten Ausgabedaten automatisch zu (*partiellen*) Datenabhängigkeitsgraphen, genannt *Ontologische Netzwerke*, verknüpft. Zur Laufzeit werden, basierend auf aktuellen Werten des Informationssystems, die bearbeitbaren Tasks aus dem Ontologischen Netzwerk abgeleitet. Mögliche Mehrdeutigkeiten, die beispielsweise durch das Bereitstellen der Eingabedaten durch unterschiedliche Prozessschritte entstehen können, werden interaktiv aufgelöst. Hier werden Mechanismen aus der künstlichen Intelligenz (Backtracking durch

Truth-Maintenance-Mechanismen [Doy79]) zur Analyse möglicher Pfade eingesetzt, um eine Entscheidungsvorlage für die Bearbeiter zu erzeugen.

Das Ontologische Netzwerk kann als Workflow-Schema betrachtet werden, bei dem nur Datenflussabhängigkeiten spezifizierbar sind. Weder Kontrollflussabhängigkeiten noch zeitliche Restriktionen können dargestellt werden. Aufgrund der nicht vorhandenen Kontrollflusskonstrukte sind somit auch keine bedingten Verzweigungen und Schleifen modellierbar, weshalb größere strukturierte Prozesse nur unzureichend dargestellt werden können. Die fehlenden Zeitrestriktionen erlauben keine Führung bei unstrukturierten Prozessteilen.

### 20.2.3 SIMNET

Betrachtet man heutige moderne Entwicklungsverbunde, so wächst die sich täglich ändernde Datenmenge, mit dem Entwickler konfrontiert werden, rapide. Somit wird es für eine Person immer schwieriger für sie relevante Änderungen aus der Flut von Informationen zu extrahieren. Häufig werden betroffene Personen – insbesondere aus nachgelagerten Entwicklungsbereichen – gar nicht oder zu spät und nur sehr grob (z. B.: „Die Getriebewelle *wurde* modifiziert“) informiert, wodurch in der Regel unnötige Kosten und Zeitverzögerungen verursacht werden. Die Verbesserung der Zusammenarbeit aller Partner eines virtuellen Entwicklungsverbunds beim Änderungsmanagement ist das erklärte Ziel des **SIMNET**-Projekts (Workflow Management for Simultaneous Engineering Nets) [RC00].

Notwendig dazu ist eine feingranulare Beschreibung von Produkten, um Änderungen präziser zu formulieren (z. B.: „Der Durchmesser der Getriebewelle wurde um 2 mm vergrößert.“) und um detaillierter zu spezifizieren, an welchen Änderungen Interesse besteht. Dazu wurden im **SIMNET**-Projekt *Parameter* als kleinste Produktdateneinheit eingeführt [GS00]. Ein Parameter beschreibt Entwicklungsattribute eines Produkts. Beispiele hierfür sind der Durchmesser oder das Material eines Bauteils. Ein auf diese Weise beschriebenes Auto umfasst zwar mehr als $10^5$ Parameterbelegungen. Für die Koordination des Änderungsmanagements sind jedoch nur circa $500$ Parameter von Bedeutung, da nur bei Veränderung einer dieser Parameter mehrere Personen oder Bereiche betroffen sind [GS00]. **SIMNET** stellt einen *Subscribe*-Mechanismus zur Verfügung, mit dessen Hilfe sich Personen bei einem Parameter registrieren können. Dies ist Voraussetzung, um bei einem Änderungswunsch bezüglich dieses Parameters informiert zu werden. Die Änderung eines Parameters beeinflusst in vielen Fällen auch andere Parameter. Diese Beziehungen lassen sich im **SIMNET**-Modell mittels *Parameternetzwerken* beschreiben.

Subscribe-Mechanismus und Parameternetzwerk bilden in Kombination mit Parameteränderungszuständen die Grundlage für partnerübergreifendes Änderungsmanagement [RC00]: Beabsichtigt ein Bearbeiter einen Parameter zu ändern, so wird der Parameter im Parameternetzwerk in den Zustand *in change* gesetzt und alle registrierten Personen informiert. Diese überprüfen den Änderungswunsch und entscheiden individuell, ob sie von der Änderung betroffen sind. Ist dies der Fall, so können sie ihre Belange einbringen oder gegebenenfalls den Änderungswunsch ablehnen. Sind sie mit dem Änderungswunsch einverstanden, so werden die Verknüpfungen des Parameternetzwerks Stufe für Stufe ausgewertet und die bei dem jeweiligen Parameter eingetragen Personen ebenfalls über den Änderungswunsch informiert, bis für jeden Pfad des Netzwerks kein „Subscriber“-Interesse besteht. Erst dann wird der Status der betroffenen Parameter von *in change* auf *approved* und nach Durchführung der Änderungen auf *released* gesetzt. Damit lässt sich auf feingranularer Ebene ein Änderungsmanagement realisieren, das nicht die oben genannten Problemfelder aufweist.

Der soeben beschriebene Genehmigungs- und Freigabe-Workflow stellt den einzigen Workflow des **SIMNET**-Systems dar. Im Gegensatz dazu bietet der **WEP**-Ansatz ein allgemeines Konzept zur Modellierung beliebiger Entwicklungs-Workflows an.

Aus **WEP**-Sicht interessant ist die **SIMNET**-Forderung nach einer feingranularen Produktstruktur und deren Analyse zur Optimierung der Prozesssteuerung. Diese Forderung, die auch von anderen Autoren [Het00, SAF+01] aufgestellt und mit Einführung von sogenannten *Features* umgesetzt wird, ist im **WEP**-Modell problemlos realisierbar. Denn die Granularität eines **WEP**-Objekts kann beliebig festgelegt werden, ohne Konflikte mit anderen **WEP**-Konstrukten zu verursachen. Ein **WEP**-Objekt kann deshalb im Sinne eines **SIMNET**-Parameters oder eines Features verwendet werden, wodurch in **WEP**-Workflows auch Datenflüsse auf Parameter- oder Feature-Granularität möglich werden.

Auch können Parameternetzwerke bei der Traversierung eingesetzt werden, um betroffene Personen beziehungsweise die von ihnen auszuführenden Prozessschritte detaillierter zu identifizieren.

Zusammenfassend können die **SIMNET**-Konzepte als Ergänzung zu den Ansätzen im **WEP**-Modell betrachtet werden. Die Autoren des **SIMNET**-Projekts sehen ihre Konzepte auch nicht als Ersatz, sondern als Ergänzung zu der dokumentenorientierten Workflow-Steuerung herkömmlicher Produktdatenmanagementsysteme [GS00].

## 20.3 Groupware-orientierte Ansätze

Bei groupware-orientierten Ansätzen [EGR91, BS98] liegt der Schwerpunkt in der Bereitstellung (synchroner) Kooperationsmechanismen. Im Folgenden werden zwei Modelle vorgestellt, die speziell für die Anforderungen von Entwicklungsprojekten konzipiert wurden.

### 20.3.1 TOGA

Entwicklungsprojekte sind geprägt durch das Management komplexer und heterogener Datenstrukturen proprietärer CAE-Applikationen, die einen Datenzugriff meist nur über vorgegebene Schnittstellen erlauben. Diese Datenkapselung schließt damit eine direkte Synchronisation kooperierender Applikationen auf Datenebene aus [HH00]. Deshalb wird beim **TOGA**-Ansatz (Transaction-Oriented Group and Coordination Service for Data-Centric Applications) [FSM00] für die kooperierende Applikationen eine Integration auf funktionaler Ebene angestrebt: Stößt eine Applikation eine Operation an, so wird diese über einen **TOGA**-Server, der alle auf dem gleichen Datenbereich arbeitende Applikationen verwaltet, an alle anderen Anwendungen weitergeleitet, die der Operation über ein 2-Phasen-Commit-Protokoll zustimmen oder diese ablehnen.

Da sich **TOGA** ausschließlich auf synchrone Kooperationsmechanismen beschränkt, ist eine Diskussion bezüglich der Umsetzbarkeit der Anforderungen aus Teil I nicht sinnvoll. Im Rahmen des **WEP**-Projekts können die **TOGA**-Konzepte bei der Unterstützung von Gruppenaktivitäten (vergleiche Teil V) und bei Konsolidierungsrunden Verwendung finden, die beide synchrone Kooperationsmuster beinhalten.

### 20.3.2 ASCEND

Zielsetzung des **ASCEND**-Ansatzes [Fra99, FM01b, FM01a] ist eine *flexible* Unterstützung der beteiligten Personen bei der *integrierten* Produktentwicklung. Schwerpunkt des Konzepts ist die Bereitstellung eines gemeinsamen *Informationsraums* zur Kooperationsunterstützung aller Beteiligten. Der Zugriff auf die Objekte des Informationsraums wird über frei definierbare Zugriffsprotokolle geregelt, wodurch die operationale Semantik der Zugriffe flexibel an den jeweiligen Anwendungsbereich angepasst werden kann [FM01a].

Zur flexiblen Kooperationsunterstützung können *Designflows* definiert werden. Ein Designflow besteht aus (starren) *Workflow-Aktivitäten* für die (Teil-)Automatisierung von strukturierten Prozessen sowie (flexiblen) *Gruppenaktivitäten* zur Unterstützung unstrukturierter Prozesse. Es ist möglich Designflows mittels *Designflow-Aktivitäten* hierarchisch zu gliedern. Aktivitäten und Objekte des Informationsraums können über die bereits aus dem **CONCORD**-Modell (siehe Abschnitt 19.3.1) bekannten Kantentypen (Negotiation, Delegation, Usage) miteinander verbunden werden [Fra99]. Ein *Publish-Subscribe*-Mechanismus erlaubt es, dass Aktivitäten über Veränderungen im Informationsraum automatisch informiert werden.

Zur Laufzeit können bei Bedarf weitere Kanten eingefügt werden. Alle Kanten werden interaktiv angestoßen. Zusammen ermöglicht dies eine flexible Unterstützung unstrukturierter Teilprozesse. Es fehlt allerdings eine zielgerichtete Führung wie sie im **WEP**-Modell mit dem Meilenstein-Konzept realisiert wurde. Simultaneous-Engineering kann durch das Bereitstellen vorläufiger Daten im Informationsraum einfach unterstützt werden. Konzepte zur Beschreibung von Datenqualitäten sind nicht vorhanden.

Für eine adäquate Modellierung strukturierter Prozesse ist das **ASCEND**-Modell nicht ausreichend, da Kontrollflüsse nicht darstellbar sind. Bedingte Verzweigungen oder Schleifen müssen „von Hand“ durch entscheidungsbevollmächtigte Aktivitätenbearbeiter ausgeführt werden. Auf die gleiche Weise wird über die Terminierung von Designflows entschieden. Den Bearbeitern werden dadurch sehr viele automatisierbare Prozesskoordinationsaufgaben aufgebürdet. Eine korrekte Prozessdurchführung und -terminierung hängt somit vom Wissen des Einzelnen über die Prozesslogik ab. Eine vorab durchführbare Korrektheitsanalyse des Designflows ist damit nicht möglich.

Insgesamt ist der **ASCEND**-Ansatz stärker für kleine überschaubare und somit auch noch durch Personen individuell koordinierbare Entwicklungsprozesse geeignet.

# Kapitel 21

# Vergleich mit Ansätzen aus dem Projektmanagement

Primäre Aufgabe des Projektmanagements ist die Planung und Überwachung von Projekten entsprechend der vorgegebenen Projektziele. Unter Planung versteht man dabei nicht nur die zeitliche Planung der verschiedenen Projektphasen und -schritte, sondern auch eine Kapazitätsplanung von Personen und sonstigen Ressourcen. Besonders bei langdauernden Projekten, wie beispielsweise bei den Entwicklungen komplexer Produkte, ist eine vollständige Projektplanung vor der eigentlichen Projektdurchführung nicht realistisch. Zu Beginn solcher Projekte werden deshalb meist nur die wichtigsten Projektphasen grob geplant. Die jeweilige Detaillierung erfolgt erst während der Projektdurchführung. Praxistaugliche Projektmanagementkonzepte müssen deshalb sowohl diese inkrementelle Projektplanung als auch Anpassungen des Projektplans während dessen Durchführung unterstützen.

Aufgrund dieser notwendigen Verschränkung von Planung und Ausführung ist es wichtig Planungskonzepte und Prozesssteuerungskonzepte zu verknüpfen. Die folgende Diskussion der Ansätze konzentriert sich deshalb im Wesentlichen auf solche Projektmanagementmodelle, die alle erwähnten Aufgaben unterstützen.

Aus informationstechnischer Sicht gibt es viele unterschiedliche Gebiete, die Lösungskonzepte für das Projektmanagement bereitstellen. Neben klassischen Netzplantechniken [NM93] existieren unter anderem Ansätze aus der künstlichen Intelligenz [LC93, MP95, Gol96] oder der temporalen Logiken [MP92]. Es wurden auch temporale Erweiterungen von Petrinetzen [RH80, Obe90, KGZH95] oder Statecharts [WDM+95] vorgeschlagen.

## 21.1 Verschränkte Planung und Ausführung in Procura und MILOS

**Procura** [Gol96] und **MILOS** (Minimally Invasive Long-term Organizational Support) [MDB+00] unterstützen beide Projektplaner bei der interaktiven Projektplanerstellung. Ein Planer erstellt hierzu *top-down* eine mehr oder weniger detaillierte hierarchische Dekomposition der durchzuführenden Aufgaben eines Projekts. Für jede Aufgabe müssen geschätzte Bearbeitungsdauer, benötigte Ein- und

Ausgabedaten[1] sowie Agenten (menschliche Bearbeiter oder Computerprogramme) spezifiziert werden. Basierend auf den Beziehungen zwischen den Ein- und Ausgabeparametern der einzelnen Aufgaben und der Verfügbarkeit der Agenten können beide Systeme automatisch einen ersten Projektplan nach der kritischen Pfad-Methode (siehe zum Beispiel [NM93]) erstellen, der durch den Planer anschließend adaptiert werden kann.

Bei beiden Ansätzen werden die einzelnen Aufgaben den spezifizierten Agenten zur Ausführung gegeben. Eine Aufgabe kann entweder als weiterer *Planungsschritt*, der sie in detailliertere Subaufgaben aufteilt und für diese einen Subprojektplan erstellt, oder als *Bearbeitungsschritt* ausgeführt werden. Ein Bearbeitungsschritt belegt die Ausgabeparameter der Aufgabe mit aktuellen Werten. **MILOS** unterstützt in der Ausführungsphase die Bearbeiter in Form von *To-do*-Listen, automatischer Bereitstellung der im Plan als Referenzen spezifizierten Eingabedaten und durch Aufrufen der benötigten Applikationen zum Erzeugen der Ausgabeparameter. Durch das Nebeneinander von Planungs- und Bearbeitungsschritten wird eine Verschränkung von Planung und Ausführung erreicht und damit die für große Projekte notwendige inkrementelle Feinplanung ermöglicht.

Beide Systeme erlauben das Verändern der Pläne zur Laufzeit. Neben Zuordnung anderer Agenten und zeitlichen Verschiebungen sind auch das Einfügen und Löschen von Aufgaben möglich. Letztere Modifikationen können aufgrund der Datenflussabhängigkeiten zu erheblichen Veränderungen des Projektplans führen. Mittels eines zugrundeliegenden *Truth-Maintenance-Systems* [Doy79, Pet93] ist es möglich auf frühere Pläne zurückzugreifen, wenn die modifizierten Pläne keinen Erfolg versprechen. Die Unterstützung der Systeme zur Sicherung der Datenkonsistenz beschränkt sich hierbei auf das Informieren der betroffenen Agenten über die Planmodifikationen. Es liegt deshalb in der Verantwortung der Planer nur „vernünftige“ Modifikationen durchzuführen.

Die Strukturierung der Prozesse erfolgt ausschließlich anhand von Datenabhängigkeiten. Rücksprünge können prinzipbedingt nicht durch Schleifen, sondern müssen durch Neuplanung zur Laufzeit umgesetzt werden.

Insgesamt lässt sich sagen, dass die Stärken beider Ansätze in der inkrementellen Planung von Projekten liegen. Die aktive Prozesssteuerung erfolgt auf einer abstrakten Ebene und beschränkt sich auf grobe Datenflussabhängigkeiten. Es sind weder komplex strukturierte Kontrollflussabhängigkeiten darstellbar noch lassen sich Ansätze zur Unterstützung von Simultaneous-Engineering oder Concurrent-Engineering finden (vergleiche **WEP**-Konzepte zur vorzeitigen Datenweitergabe beziehungsweise zur Objekttraversierung). Die geforderte zeitliche Steuerung unstrukturierter Teilprozesse ist über geeignete Zeitvorgaben und Datenabhängigkeiten realisierbar.

Die Idee der Verschränkung von Planung und Ausführung auf der Basis von künstlichen Intelligenz-Methoden wurde in ähnlicher Form auch von **EPOS** [LC93, NC96] und **CoMo-Kit** [MP95, Mau96, DMP97] aufgegriffen. Beide Ansätze beschränken sich jedoch auf *relative* zeitliche Beziehungen zwischen den verschiedenen Aufgaben, die sich aufgrund von Abhängigkeiten ihrer Ein- und Ausgabeparameter ergeben. *Absolute* zeitliche Restriktionen durch den Einsatz von Netzplantechniken fehlen dagegen.

[1] Bei **MILOS** sind diese aus einem *Produktmodell* selektierbar, das Datentypen und damit assoziierte Applikationen beschreibt.

## 21.2 Zeitplanung von Workflow-Instanzen

Bei den in Abschnitt 21.1 diskutierten Ansätzen wurden Projektplanungskonzepte erweitert, um Planung *und* Ausführung von Prozessen zu ermöglichen. Dagegen wird in [Bus98] eine engere Kopplung von Workflow-Management-Systemen und Projektmanagementsystemen gefordert, um nicht bereits vorhandene Funktionalität des einen Systems im anderen System erneut implementieren zu müssen. Der Autor schlägt eine Kopplung auf Server-Ebene vor: Beim Start einer Workflow-Instanz informiert die Workflow-Runtime-Komponente das Projektmanagementsystem. Dieses erstellt für die Workflow-Instanz einen Projektplan, der die Verfügbarkeit von Ressourcen für die einzelnen Workflow-Schritte sichert und dem Workflow-Management-System absolute Zeitvorgaben für das Anbieten der Workflow-Schritte berechnet. Auf diese Weise können die Bearbeitungszeiten einer Workflow-Instanz vorausbestimmt und Ressourcen-Engpässen durch Analysen des Projektmanagementsystems vorgebeugt werden.

Diese Vorgehensweise liegt auch dem Zeitmanagement des **WEP**-Workflow-Management-Systems zugrunde [Kos00]. Auch hier wird eine Zeitabschätzung für die einzelnen Workflow-Schritte wie auch für den gesamten **WEP**-Workflow versucht. Das im **WEP**-Modell verwendete MPM-Verfahren[2] (siehe zum Beispiel [NM93]) wurde dahingehend erweitert, um auch Workflows mit Schleifen und bedingten Verzweigungen zeitlich planen zu können. Dies ist in den Konzepten aus [Bus98] nicht vorgesehen. Die dort als *fortlaufendes Planen* bezeichnete Vorgehensweise plant vorab immer nur bis zur ersten Verzweigung[3]. Erst nachdem das Verzweigungsergebnis feststeht, wird mit der Planung des gewählten Zweigs fortgefahren. Nachteilig hierbei ist, dass dadurch keine Abschätzung für die Gesamtlaufzeit eines Workflows mit Verzweigungen möglich ist.

Natürlich wird auch beim **WEP**-Ansatz das fortlaufende Planen praktiziert, um einerseits den initial erstellten Plan den realen Bearbeitungszeiten anzupassen und anderseits, um auf externe Veränderungen der Meilensteinvorgaben reagieren zu können. Letzteres ermöglicht den Projektplanern, die Auswirkungen ihrer Planänderungen sichtbar zu machen.

## 21.3 iViP: Framework für adaptive Prozessplanung und -ausführung

Ein weiteres Projektmanagementkonzept wurde im Rahmen des **iViP**-Projekts (Integrated Virtual Product Creation) entworfen [EKH+01, KTA02]. Das unter anderem von mehreren deutschen Automobilherstellern und -zulieferern getragene Projekt stellt eine komponentenbasierte Integrationsplattform zur Verfügung, die neben vielen anderen Komponenten auch das adaptive Projektplanungs- und -steuerungsmodul **APM** (Adaptiver Prozessmanager) [EHDN00, ENK+02] beinhaltet, das im Folgenden näher betrachtet werden soll:

Das **APM** besticht weniger durch neue intelligente Projektplanungskonzepte – hier bieten die durch künstliche Intelligenz gestützten teilautomatischen Planer **Procura** und **MILOS** (siehe Abschnitt 21.1) die interessanteren Ansätze, sondern in der Bereitstellung einer Framework-Architektur, die eine kontinuierliche Rückkopplung zwischen *Soll-Projektplan* (Planungsprozess) und den *operativen Prozessen* im Produktentstehungsprozess ermöglicht (vergleiche Abbildung 21.1).

[2] Metra Potential Method

[3] Hier ist mit Verzweigung der Beginn einer *bedingten Verzweigung* oder einer *Schleife* gemeint.

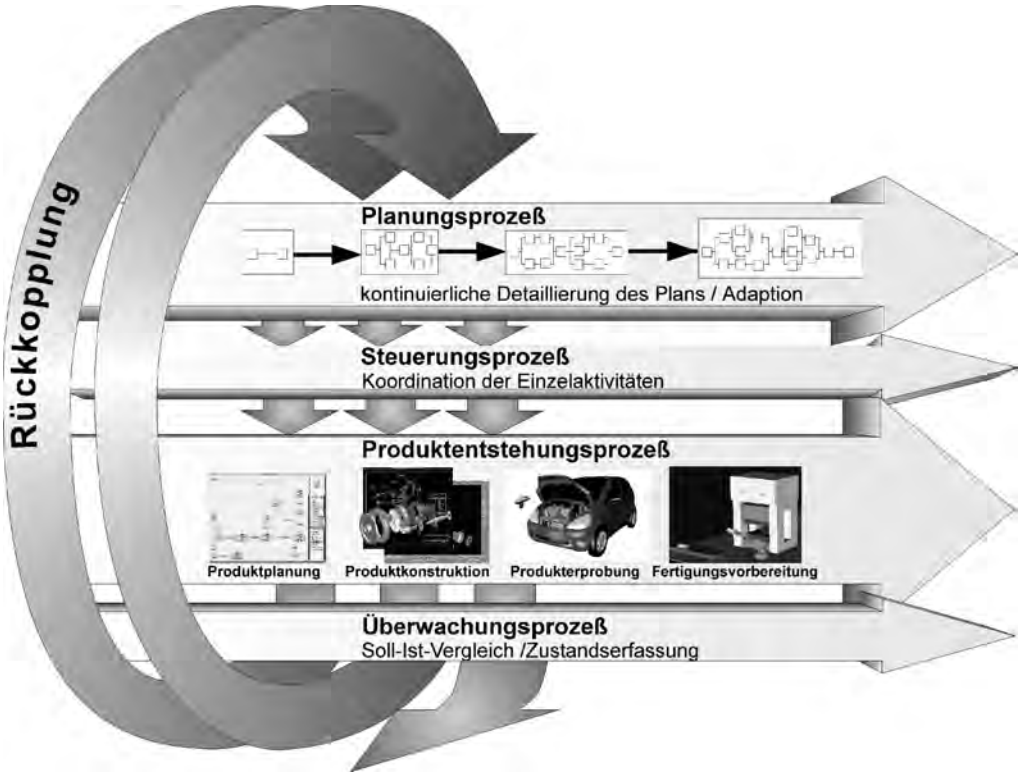



**Abbildung 21.1:** Das **iViP**-Framework (aus [ENK+02])

Ausgangspunkt ist ein grober Projektplan (in Abbildung 21.1 oben), den ein Planer im Projektverlauf kontinuierlich detailliert beziehungsweise adaptiert. Die dafür notwendigen Informationen, sogenannte *Effekte*, erhält er von *Prozesssteuerungssystemen*, welche die operativen Aktivitäten der Produktentwicklungsprozesse koordinieren. Die Bandbreite von Prozesssteuerungssystemen reicht von einfachen Applikationen zur Bearbeitung elementarer Prozessschritte bis vollständigen Workflow-Management-Systemen, die Teilprozesse des Plans als operative Entwicklungs-Workflows selbständig abarbeiten.

Um im letzteren Fall eine kontinuierliche Rückkopplung aus den (langdauernden) Entwicklungs-Workflows zu ermöglichen, werden je nach Funktionalität des beauftragten Workflow-Management-Systems Zwischenzustände der Workflow-Instanzen in der Überwachungsprozessebene gesammelt und als Effekte an die Projektplaner zurückgemeldet.

Neben dieser kontinuierlichen Rückkopplung bietet dieser Ansatz vielfältige Konfigurationsmöglichkeiten. So wurde beispielsweise nachgewiesen, dass das **WEP**-Workflow-Management-System aufgrund seiner Flexibilität große Planungsaufgaben als *ein* **WEP**-Workflow abarbeiten kann, wodurch sich die eigentlichen Planungsprozesse auf wenige generische und damit wiederverwendbare Planungsschemata reduzieren lassen [KTA02]. Außerdem zeigte sich, dass die Rückmeldung von Effekten, die auf vorläufig weitergereichten **WEP**-Ergebnissen beruhten, für eine effiziente Vorausplanung der nächsten Schritte sehr nützlich ist. Im diesen Sinne stellt das **iViP**-Projekt eine Bestätigung für die entwickelten **WEP**-Konzepte dar.

## 21.4 Weitere Projektplanungsansätze

Eine andere bisher noch nicht diskutierte Form des Projektmanagements wird in [JH95] vorgeschlagen. Hier wird nicht nur eine Hierarchie von zu erledigenden Aufgaben erstellt, sondern auch eine Dekomposition von zu erreichenden Projektzielen, die den einzelnen Aufgaben zugewiesen werden. Diese Vorgehensweise verspricht nach Ansicht der Autoren wichtige Vorteile: Explizite Ziele sind für alle Projektbeteiligten klarer, sie erlauben gefällte Entscheidungen nachzuvollziehen und durchgeführte Aufgaben können anhand ihrer Zielvereinbarung (automatisch) validiert werden.

Nachteilig ist jedoch, dass Abhängigkeiten nur durch die hierarchische Gliederung formulierbar sind. Andere Aspekte (z. B. zeitliche oder Vorgänger-Nachfolger-Restriktionen) sind nicht oder nur sehr umständlich darstellbar, so dass dieser Ansatz nur für die *Planung kleiner* Projekte vernünftig einsetzbar ist. Für eine aktive Steuerung von Produktentwicklungsprozessen ist dieses Modell deshalb nicht geeignet.

In [CM95] wird durch einen auf abstraktem Niveau geführten Vergleich von Projekt- und Workflow-Management-Systemen aufgezeigt, dass viele gemeinsame Aufgaben für beide Systeme existieren. Die nach Ansicht der Autoren künstliche Trennung führt in der Praxis zu Zusatzaufwand und Inkonsistenzen, da Informationen mehrmals erfasst werden, die bei Änderungen im einen System nicht mehr im anderen System aktualisiert werden. Es wird deshalb ein System gefordert, das die Funktionalität für Projekt- und Workflow-Management bereitstellt. Dabei wird allerdings übersehen, dass zwar die gleiche Funktionalität benötigt wird, diese jedoch aufgrund der unterschiedlichen Abstraktionsniveaus in verschiedener Ausprägung. Ein solcher Ansatz stellt deshalb zumindest sehr hohe Anforderungen an die Oberflächengestaltung und die Benutzerführung im System. Eine Kopplung auf Informationsebene, beispielsweise durch einen kontinuierlichen Informationsaustausch, wie im **iViP**-**APM**-Projekt vorgeschlagen (siehe Abschnitt 21.3), erscheint deshalb für den praktischen Einsatz vielversprechender. In der Praxis wird man auch häufig damit konfrontiert sein, dass das eine oder andere System bereits eingesetzt wird. Eine Einführung eines völlig neuen Systems mit umfangreicher Funktionalität steht deshalb aus Kosten- und Akzeptanzgründen meist nicht zur Disposition.

# Kapitel 22

# Zusammenfassung der Diskussion wissenschaftlicher Ansätze

Die Diskussion verwandter Ansätze hat nachgewiesen, dass die bisherigen Vorschläge nicht alle Praxisanforderungen aus Teil I adäquat unterstützen können. Die wichtigsten Diskussionsergebnisse werden in Abbildung 22.1 zusammengefasst.

Wie schon auf den ersten Blick erkennbar werden dabei die prozessorientierten Ansätze den Anforderungen noch am ehesten gerecht: Hier ist die Koordination des *strukturierten* Anteils von Entwicklungsprozessen gewährleistet. Es fehlen jedoch Konzepte zur Einbettung und Koordination *unstrukturierter* Teilprozesse. Nur beim **INCOME**-Ansatz sind adäquate Konzepte für automatische Reaktionen auf Veränderung der Produktstruktur vorhanden. Auch verhindert das einfache Ausführungsmodell, auf dem die Ansätze in der Regel basieren, meist eine geeignete Unterstützung von Simultaneous-Engineering-Phasen.

Die datenzentrierten Ansätze und die Projektmanagement-Konzepte behandeln nur Teilaspekte der Praxisanforderungen (siehe Abbildung 22.1). Bei den Datenmanagementmodellen werden Projektmanagementanforderungen nicht berücksichtigt. Die aktive Prozesskoordination beschränkt sich auf einfache Koordinationsprimitive, auf ein bereitgestelltes, aber nicht erweiterbares Workflow-Schema (**SIMNET**) oder auf Konzepte, die einen hohen administrativen Aufwand bei den Prozessverantwortlichen erfordern (**ASCEND**). Die Stärken der datenorientierten Ansätze liegen in der Verwaltung komplexer Datenstrukturen und ihrer Konsistenzsicherung.

Der Bezug auf Datenstrukturen fehlt dagegen bei den Vorschlägen aus dem Projektmanagementumfeld. Außerdem ist die Ausdrucksmächtigkeit zur Beschreibung von Prozessstrukturen zu gering, da meist nur grobe Datenabhängigkeiten betrachtet werden. Die Stärken dieser Ansätze liegen in der Zeit- und Ressourcenplanung auf abstrakter Prozessebene.

| | **prozessorientiert** | | | | **datenorientiert** | | | | | **projektorientiert** | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **AHEAD, DYNAMITE** | **INCOME** | **CONCORD** | **ADEPT** | **IMLE** | **Ontologische Netzwerke** | **SIMNET** | **TOGA** | **ASCEND** | **Procura, MILOS** | **CoMo-Kit, EPOS** | **Zeitplanung in Workflows** | **iViP-APM** |
| Unterstützung unstrukturierter Teilprozesse in strukturierten Prozessen | ○[1] | ○[2] | −[3] | ○[4] | | ○[5] | −[6] | −[7] | −[8] | −[9] | −[9] | | −[10] |
| Prozesskoordiniertes Simultaneous-Engineering | + | −[11] | + | −[11] | | | ○[12] | ○[13] | ○[14] | | | | |
| Dynamische Anpassung einer Prozessstruktur | ○[15] | + | | ○[16] | −[17] | | ○[18] | | | | | | ○[19] |
| Abstimmung Prozesssteuerung mit adaptiver Projektplanung | | | | ○[20] | | | | | | + | ○[21] | ○[22] | ○[21] |
| Sicherstellung einer korrekten Prozesssteuerung | −[23] | ○[24] | −[25] | ○[24] | −[26] | | | | | | | | |

**Erklärungen:** (Bei Feldern ohne Eintrag wurden die Anforderungen nicht im Ansatz berücksichtigt)
− keine beziehungsweise unzureichende Unterstützung
○ Umsetzung mit Einschränkungen möglich
+ adäquate Umsetzung realisierbar

[1] keine Steuerung unstrukturierter Teilprozesse
[2] nur strukturierte Prozesse
[3] keine Vormodellierung auf Administrations-/Kooperationsebene
[4] unstrukturierte Teilprozesse durch manuellen Graphumbau
[5] nur strukturierte Prozesse, nur Datenflussabhängigkeiten
[6] nur ein Änderungs-Workflow, keine Steuerung unstrukturierter Teilprozesse
[7] nur unstrukturierte Teilprozesse (als Gruppenaktivitäten), keine asynchronen Synchronisationsmechanismen
[8] keine automatische übergeordnete Prozesskoordination
[9] nur Prozesssequenzen, nur Datenflussabhängigkeiten
[10] nur abstrakte Prozesssequenzen, nur Datenflussabhängigkeiten
[11] einfaches Ausführungsmodell nicht ausreichend für Simultaneous-Engineering
[12] publish-subscribe-Mechanismus zum Informieren interessierter Personen
[13] einfacher Trigger-Mechanismus
[14] Simultaneous-Engineering innerhalb von Gruppenaktivitäten
[15] manuelle Anpassungen möglich
[16] nicht im Modell enthalten, über automatische Änderungsoperationen möglich
[17] keine Datenqualitäten, nur einfacher Trigger-Mechanismus
[18] Traversierung mittels Parameternetzwerk
[19] manuelle Anpassungen durch Planer
[20] fehlende zeitliche Führung innerhalb unstrukturierter Teilprozesse
[21] nur relative Zeitvorgaben
[22] keine Zeitabschätzungen bei Workflows mit Schleifen oder bedingten Verzweigungen
[23] keine Zusicherung der Blockierungsfreiheit nach dynamischen Umbau
[24] Korrektheit nur für einfaches Ausführungsmodell
[25] Korrektheit nur auf Werkzeugebene
[26] nur Datenkorrektheit, keine Prozesskorrektheit

**Abbildung 22.1:** Vergleich der wichtigsten Lösungsansätze mit den Praxisanforderungen aus Teil I

# Kapitel 23

# Vergleich mit kommerziellen Systemen

Wurde in den vorhergehenden Kapiteln dieses Teils der Stand der Wissenschaft diskutiert, so skizziert dieses Kapitel aus dem Blickwinkel der Prozesskoordinationsanforderungen den Stand der heute in der Produktentwicklung eingesetzten beziehungsweise kommerziell verfügbaren Technologien.

### State of practice

Wie schon in der Einleitung erwähnt, ist die Funktionalität heutiger kommerziell verfügbarer Systeme, die im Umfeld der Produktentwicklung eingesetzt werden, bei weitem nicht ausreichend, um die Praxisanforderungen umzusetzen [HB97, BKJ+99]. In den Entwicklungsbereichen werden zum Management der anfallenden Entwicklungsdaten heutzutage meist mainframe-basierte Eigenentwicklungen eingesetzt, die keinerlei Prozesssteuerungsfunktionalität und nur sehr begrenzte Möglichkeiten zur Produktstrukturierung besitzen. In den letzten Jahren kommen langsam unix-basierte Produktdatenmanagementsysteme verschiedener Hersteller (z. B.: **Metaphase** [SDRC99] von EDS (früher SDRC), **VPM** [IBM99a], **Enovia R5** [DS02] von Dassault-Systemes, **Windchill** [PTC01a, PTC01b] von PTC, **eMatrix** [Matrix01a, Matrix01b, Matrix01c] von Matrix One) zum Einsatz, die in der Regel auch eine Workflow-Komponente beinhalten. Diese *eingebetteten* Workflow-Management-Systeme verwenden die ausgefeilte Daten- und Versionsmanagementfunktionalität der sie umschließenden Produktdatenmanagementsysteme. Dies ist *externen* Workflow-Management-Systemen so nicht möglich, da sie aufgrund ihrer stärkeren allgemeinen Ausrichtung meist auf einem sehr viel einfacheren Datenmodell aufsetzen. Dieses Defizit ist sicherlich einer der Gründe, weshalb externe Workflow-Management-Systeme derzeit nicht zur Steuerung von Prozessen in Entwicklungsbereichen eingesetzt werden.

### State of technology

Anhand der Praxisanforderungen aus Teil I der Arbeit wird kurz die Eignung der derzeit kommerziell verfügbaren Workflow-Management-Technologien zur Unterstützung praxisrelevanter Produktentwicklungsprozesse diskutiert, um dem Leser den „*state of technology*“ kommerzieller Systeme zu skizzieren.

*Unterstützung unstrukturierter Teilprozesse innerhalb vorgegebener Prozessstrukturen:*

Die Workflow-Konzepte der wichtigsten Produktdatenmanagementsysteme beinhalten zum Teil nur einfache Trigger-Mechanismen (**VPM**, **eMatrix**), die ungeeignet sind, komplexe

Prozesse adäquat zu koordinieren. Weitergehende Konzepte (**Metaphase**, **Enovia R5**, **Windchill**) erlauben nur die Modellierung und Steuerung *starrer* Prozesse. Sie sind somit nicht in der Lage, moderne flexible Entwicklungsprozesse zu koordinieren, sondern eignen sie allenfalls für periphere administrative Prozesse, wie beispielsweise Freigabeprozesse[1].

Kommerzielle externe Workflow-Management-Systeme besitzen ebenfalls nur die Modellierungsmächtigkeit, um starre Prozesse geeignet zu modellieren und zu koordinieren. Die Einbettung unstrukturierter Teilprozesse wird bei kommerziellen Systemen entweder gar nicht unterstützt oder ist nicht befriedigend gelöst: **MQSeries Workflow** [IBM99b, IBM01a, IBM01b] ermöglicht es zwar, Prozessschritte an das Groupware-System **Lotus Notes** [Lotus99, IBM01c] zu transferieren und dort überwachen zu lassen. Neben erhöhtem Administrationsaufwand durch doppelte Benutzerverwaltung und nur sehr rudimentären Möglichkeiten zum Datenaustausch stehen **MQSeries Workflow** keinerlei Statusinformationen über den Verlauf der Bearbeitung im Groupware-System zur Verfügung. **InConcert** [MS93, TIBCO01] erlaubt es mittels *leerer* Workflow-Vorlagen Bearbeitungsschritte in einem Gesamt-Workflow ad hoc zu definieren, um damit unstrukturierte Teilprozesse abzubilden. In beiden Ansätzen fehlt jedoch jegliche Möglichkeit zur Führung der Bearbeiter durch einen unstrukturierten Teilprozess. Die Datenversorgung nachfolgender Schritte kann nicht gewährleistet werden. Auch können die Programmschritte eines unstrukturierten Teilprozesses nicht auf eine sinnvolle Auswahl zur Laufzeit beschränkt werden. Es obliegt damit eine größere Verantwortung und ein höherer administrativer Aufwand bei den Bearbeitern unstrukturierter Teilprozesse, ohne wirklich praktischen Nutzen daraus ziehen zu können.

*Prozesskoordiniertes Simultaneous-Engineering:*

Alle kommerziellen Systeme mit Workflow-Management-Funktionalität basieren auf dem einfachen Workflow-Ausführungsmodell, bei dem Ergebnisse erst mit Ende eines Workflow-Schritts an Folgeaktivitäten weitergegeben werden. Dies schließt ein prozesskoordiniertes Simultaneous-Engineering durch frühzeitige Datenweitergabe vorläufiger Daten aus. Moderne Produktdatenmanagementsysteme bieten in der Regel Möglichkeiten, verschiedene Bearbeitungszustände von Objekten und erlaubte Übergangsbedingungen zu spezifizieren (*Lifecycle-Management*). Damit lassen sich – wenn auch auf abstraktem Niveau – Datenqualitätsstufen darstellen. Diese Datenqualitätsstufen finden aber bei der Modellierung des Datenflusses in der Workflow-Komponente des jeweiligen Produktdatenmanagementsystems keine Verwendung. Somit fehlen bereits die Modellierungsmöglichkeiten zur Beschreibung vorzeitiger Datenweitergaben in Workflows.

In **Enovia R5** wurden in vereinfachter Form die in **IMLE** (vergleiche Abschnitt 20.2.1) und **SIMNET** (vergleiche Abschnitt 20.2.3) vorgeschlagenen Konzepte zur Beschreibung von Abhängigkeiten zwischen Objekten realisiert. Dort ist es möglich, Konsistenzabhängigkeiten zwischen verschiedenen Objekten zu spezifizieren, die gemeinsam ein (Teil-)Produkt beschreiben. Ändert sich die Version eines der Objekte, so müssen die anderen Objektversionen eventuell angepasst werden, um Inkonsistenzen zu vermeiden. Deshalb werden zur Laufzeit simultan arbeitende Benutzern auf Nachfrage über solche potenziellen Inkonsistenzen informiert (*Impact-Mechanismus)*.

[1] Freigabeprozesse legen die administrativen Schritte fest, die *nach* der Durchführung von konstruktiven Änderungen an Fahrzeugkomponenten abgearbeitet werden müssen, um diese für die Produktion freizugeben.

Die Unterstützung von Ad-hoc-Interaktionen beschränkt sich bei Produktdatenmanagementsystemen in der Regel auf die Bereitstellung eines *gemeinsamen Datenbereichs* und die Synchronisation des Zugriffs mittels *Check-in/out*-Funktionalität sowie einfache *Email*- und *Subscribe*-Mechanismen.

Die Grundlagen zur Unterstützung von Simultaneous-Engineering sind damit zumindest rudimentär vorhanden. Es fehlt jedoch die Einbindung dieser Konzepte in die Workflow-Koordination.

Externe Workflow-Management-Systeme erlauben in beschränktem Maße Ad-hoc-Interaktionen mit anderen am Workflow beteiligten Personen. Hierunter fallen Rückfragen bei unvollständigen oder missverständlichen Eingabedaten (**ProMInanD** [Kar94, IABG98a, IABG98b]), das Zurückweisen beziehungsweise Delegieren von Aufgaben oder auch einfache strukturelle Veränderungen (Einfügen, Verschieben, Überspringen, Löschen von Workflow-Schritten) (**ProMInanD**, **Workflow** [JBS97] oder **InConcert** [MS93, TIBCO01]), wobei die Korrektheit der Modifikationen ausschließlich in der Verantwortung der Bearbeiter liegt und somit ein hohes Risikopotenzial für Inkonsistenzen und fehlerhafte Prozessdurchführung darstellt.

*Dynamische Anpassung einer vorgegebenen Prozessstruktur:*
Mit Ausnahme des Produktdatenmanagementsystems **Metaphase**, bei dem mit großen Einschränkungen mittels *Breakdown*-Prozessen abhängig von der aktuellen Objektstruktur dynamisch parallele Schritte generiert werden können, sind *automatische* Prozessanpassungen im kommerziellen Systemen nicht möglich. Breakdown-Prozesse können jedoch nur auf nicht veränderbare Objekte angewandt werden. Auch können nur die direkten Kinder des auslösenden Objekts an die dynamisch erzeugten parallelen Schritte weitergereicht werden (nur einstufige Relationsverfolgung).

*Abstimmung Prozesssteuerung mit adaptiver Projektplanung:*
Produktdatenmanagementsysteme – wie auch vereinzelt Workflow-Management-Systeme (z. B.: **MQSeries Workflow**, **Staffware** [Sta02b, Sta02a, Sta02c]) – ermöglichen die Spezifikation von Zeitrestriktionen und ihre Zuordnung zu Prozessschritten. Überschreitungen lösen vordefinierte Aktionen aus. Es fehlen allerdings Möglichkeiten zur Bestimmung kritischer Pfade und zur Berechnung der Auswirkungen bei Veränderungen von Zeitrestriktionen. Wie auch bei externen Workflow-Management-Systemen beziehen sich die Meilensteinvorgaben auf Start- und Ende von Prozessschritten und nicht auf die Bereitstellungstermine von Ergebnissen bestimmter Qualität.

*Sicherstellung einer korrekten Prozesssteuerung:*
Die Validierungsmöglichkeiten heutiger Systeme beschränken sich in den meisten Fällen auf einfache Prüfungen (z. B. zusammenhängende Workflow-Graphen, Vollständigkeit bei der Zuordnung von Organisationselementen zu Workflow-Schritten oder bei der Verknüpfung von Parametern mit Datenflusskanten). Komplexere Analysen konsistenter Datenflüsse oder der Verklemmungsfreiheit ist auch bei petrinetzbasierten kommerziellen Systemen (z. B.: **INCOME** [Pro01]) nicht gegeben.

Zusammenfassend weisen kommerziell verfügbare Systeme also noch erhebliche Mängel bei der Koordination von Entwicklungsprozessen auf. Die fehlende Flexibilität und das für moderne Entwicklungsprozesse ungeeignete einfache Ausführungsmodell stellen hierbei neben der fehlenden Prozess-

sicherheit sicherlich die größten Hindernisse dar, weshalb die Unternehmen ihre Kernentwicklungsprozesse nicht durch diese Systeme steuern lassen.

# Teil V

# Zusammenfassung und Ausblick

# Kapitel 24

# Zusammenfassung der Ergebnisse

Die Globalisierung der Märkte, der E-Business-Boom sowie eine stärkere und flexiblere Kundenorientierung verschärfen die Notwendigkeit für ein Unternehmen, seine Prozesse kontinuierlich zu optimieren, flexibel an neue Anforderungen anzupassen und die Planungssicherheit sowie Kostentransparenz zu verbessern. Hierbei ist eine geeignete IT-Unterstützung unabdingbar, um die identifizierten Optimierungspotenziale voll ausschöpfen zu können, da auch in optimierten Prozessen die involvierten Personen einen hohen Anteil nicht wertschöpfender repetitiver Tätigkeiten verrichten, die man gerne Computersystemen übertragen möchte. Realistisch betrachtet sind Prozesse, die über einfache „Spielanwendungen“ hinausgehen und in großer Zahl durchgeführt werden, ohne Computereinsatz nicht vernünftig koordinierbar. Die Verbesserung der Prozesstransparenz und -qualität sind weitere Gründe für ein computergestütztes Prozessmanagement.

Die Prozessunterstützung ist die Kernaufgabe von Workflow-Management-Systemen. Im Gegensatz zu anderen IT-Systemen kann aufgrund der Trennung von Ablauflogik und eigentlichem Anwendungscode ein zu steuernder Prozess schnell spezifiziert, validiert und rasch an Prozessveränderungen angepasst werden. Workflow-Management-Systeme besitzen damit prinzipiell das Potenzial, die geforderten Wünsche nach einem computergestützten Prozessmanagement zu erfüllen und haben dies zum Teil schon in Branchen mit gut vorstrukturierbaren Prozessen, wie sie beispielsweise im Dienstleistungsbereich zu finden sind, demonstriert.

Dagegen wurden Workflow-Management-Systeme beim produzierenden Gewerbe für die dortigen Kernprozesse (Entwicklung, Produktion) bisher noch nicht eingesetzt, obwohl auch hier wiederholende Prozesse – die wichtigste Voraussetzung für den Einsatz von Workflow-Management-Technologie – anzutreffen sind. Die Gründe dafür wurden in Teil I der Arbeit ausführlich dargelegt: Anforderungen wie die Unterstützung *unstrukturierter* Teilprozesse in einem *strukturierten* Gesamtprozess, das *Simultaneous-Engineering* entlang der Prozesskette, *dynamische Anpassungen an eine veränderte Produktstruktur*, das Zusammenspiel mit einem *adaptiven Projektmanagement* und die Sicherstellung einer *korrekten Prozesssteuerung* sind essenziell für den Einsatz von Workflow-Management-Systemen in diesen Bereichen. Die fehlende Eignung für Produktentwicklungsprozesse weisen dabei nicht nur die kommerziell verfügbaren Workflow-Management-Systeme auf [HB97, BKJ+99], sondern – wie in Teil IV dieser Arbeit gezeigt – auch die in der Wissenschaft vorgeschlagenen Prozesssteuerungskonzepte.

Das Anliegen dieser Arbeit ist es deshalb gewesen, sich diesen Praxisanforderungen zu stellen und eine *adäquate Workflow-Management-Unterstützung für Produktentwicklungsprozesse* zu entwicklen. Durch die Konzeption und prototypische Implementierung des **WEP**-Workflow-Management-Systems konnten diese Ziele erfolgreich realisiert werden. Die wichtigsten Ergebnisse lassen sich dabei wie folgt zusammenfassen:

**Unterstützung unstrukturierter Teilprozesse innerhalb vorgegebener Prozessstrukturen**

Das in dieser Arbeit entwickelte **WEP**-Workflow-Management-System basiert auf einem Graphmodell mit vollständiger Blockstruktur. Nur so lassen sich komplex strukturierte Entwicklungsprozesse angemessen modellieren und warten. Dieser prozessorientierte Ansatz stößt natürlich an seine Grenzen, wenn damit auch unstrukturierte Prozesse beschrieben werden sollen, weil sich hier die Reihenfolgebeziehungen der einzelnen Prozessschritte erst zur Ausführungszeit ergeben. Es wurde gezeigt, dass für unstrukturierte Teilprozesse eine zielorientierte Beschreibung der prozessorientierten Modellierung vorzuziehen ist. Deshalb wurde das Konzept der *zielorientierten Aktivitäten* eingeführt, bei denen die Führung der Bearbeiter durch den unstrukturierten Teilprozess nicht durch die Vorgabe von Bearbeitungsreihenfolgen, sondern durch Festlegen von Zielen in Form von abzulieferenden Zwischen- und Endergebnissen bestimmter Qualität zu festgelegten Zeitpunkten erfolgt. Damit ist es – im Gegensatz zu anderen Ansätzen – gelungen, ziel- und prozessorientierte Modellierungstechniken in einem *Gesamtmodell* zu vereinen. Denn es wurde gezeigt, wie zielorientierte Aktivitäten analog zu Prozessbausteinen anderer Workflow-Management-Systeme mittels Kontroll- und Datenflüssen zu einem strukturierten Gesamt-Workflow verknüpft werden können. Mit diesen zusätzlichen Modellierungsfreiheiten kann die gesamte Bandbreite von völlig unstrukturierten Abläufen bis zu vollständig vorstrukturierbaren Prozessen abgedeckt werden. Das **WEP**-Modell stellt in diesem Sinne eine echte Obermenge herkömmlicher Workflow-Management-Technologien dar.

**Prozesskoordiniertes Simultaneous-Engineering**

Die Konzeption von Zwischen- und Endergebnissen bei zielorientierten Aktivitäten bildet auch die Basis zur Umsetzung der Anforderung der Simultaneous-Engineering-Unterstützung entlang der Prozesskette. Simultaneous-Engineering wird natürlich erst dann ermöglicht, wenn die Zwischenergebnisse bereits vor Beendigung eines Prozessschritts an Folgeschritte weitergereicht werden können. Aus diesem Grund wurde beim **WEP**-Ansatz – zum ersten Mal in einem prozessorientierten Workflow-Modell – die *Datenweitergabe von der Beendigung eines Workflow-Schritts getrennt*.

Die Folgen dieser notwendigen Maßnahme sind aber komplex:

- Zur Sicherung der Daten- und Prozesskonsistenz werden geeignete Mechanismen benötigt. Hierfür wurden in der Arbeit zum einen Modellierungsregeln für einen korrekten Datenfluss entwickelt, die unter anderem festlegen, dass Aktivitäten zwar vorläufige Daten erhalten dürfen aber auch immer mit endgültigen Daten nachversorgt werden. Zum anderen wurden *Laufzeit-Konsistenzsicherungsmaßnahmen* erarbeitet, die eine konsistente Integration neu eingetroffener *Datenversionen* in bereits in Bearbeitung befindlichen Aktivitäten erlauben. Außerdem wird durch diese Algorithmen sichergestellt, dass Workflow-Pfade, die aufgrund vorläufiger Datenweitergabe fälschlicherweise beschritten wurden, automatisch und konsistent zurückgesetzt werden.

- Simultaneous-Engineering erhöht in der Regel auch den Abstimmungsbedarf zwischen den Prozesspartnern. Ausschließlich asynchrone Kooperationsformen reichen hierbei nicht aus. In der Arbeit wurde deshalb aufgezeigt, wie auch *synchrone Interaktionsformen* in das **WEP**-Modell *integriert* werden können, um auch ad hoc entstandenen Abstimmungsbedarf zu unterstützen. Insgesamt sind die Interaktionen des Anwenders mit dem **WEP**-Workflow-Management-System weitaus komplexer. Die Benutzerinteraktionen wurden in der Arbeit spezifiziert und die erlaubten Interaktionsmuster dargestellt.
- Simultaneous-Engineering erfordert die zeitliche Verschränkung sequentiell modellierter Prozessschritte und -blöcke. Ein solches Laufzeitverhalten ist mit dem in den Workflow-Ansätzen überlicherweise verwendeten einfachen Ausführungsmodell nicht möglich, das im Wesentlichen auf einem *einmaligen* Schalten jedes Konstrukts im Workflow-Graphen beruht. Durch Simultaneous-Engineering müssen die Workflow-Konstrukte potenziell mehrmals schalten können. Es musste deshalb ein neues und weitaus komplexeres Schaltverhalten für jedes Workflow-Konstrukt für das **WEP**-Modell spezifiziert werden.

**Sicherstellung einer korrekten Prozesssteuerung**

Natürlich muss bei so tiefgreifenden Veränderungen der Ausführungslogik eines Workflow-Management-Systems die Korrektheit des neuen Schaltverhaltens garantiert werden. Basierend auf einer *formalen Beschreibung* des **WEP**-Ausführungsmodells und dem neu definierten Korrektheitsbegriff der *SE-Serialisierbarkeit* wurde in der Arbeit ein solcher Korrektheitsnachweis erbracht.

**Dynamische Anpassung einer vorgegebenen Prozessstruktur**

Die Herausforderung bestand hierbei in der Entwicklung eines Verfahrens, das basierend auf einer Eigenschaftsbeschreibung geeignete Objektinstanzen selbständig aus einer komplexen Objektstruktur selektiert und den Workflow-Graphen *automatisch* umbaut. Die bisherigen flexiblen Workflow-Konzepte erfordern dagegen in der Regel immer Interaktionen mit den Benutzern. Der entworfene *Traversierungsalgorithmus* bereinigt auch automatisch Inkonsistenzen, die durch mehrmaliges Weitergeben von Daten und damit wiederholtes Anstoßen der Traversierung entstehen können.

**Abstimmung Prozesssteuerung mit adaptiver Projektplanung**

Das *Meilensteinkonzept* der zielorientierten Aktivitäten erleichtert auch die Kopplung von **WEP**-Workflows mit einer adaptiven Projektplanung. Im Rahmen einer zu dieser Arbeit parallel entstandenen Diplomarbeit wurde außerdem gezeigt, wie die Veränderungen der Zeitvorgaben aus einer übergeordneten Projektplanung in laufende Workflow-Instanzen integriert werden können.

Ein weiteres wichtiges Resultat dieser Arbeit stellt die Tatsache dar, dass die entwickelten *Einzelkonzepte* zu einer *adäquaten Gesamtlösung* zur Modellierung und Steuerung von Entwicklungsprozessen *integriert* wurden. Andere wissenschaftliche Ansätze konnten dagegen nur Teile der Praxisanforderungen umsetzen.

Auch wurden die entwickelten **WEP**-Konzepte prototypisch umgesetzt [Sau99, Kno99, Sch00, WEP00, WEP01], um einerseits ihre Machbarkeit nachzuweisen und anderseits den Nutzen für die an

Entwicklungsprozessen beteiligten Personen zu demonstrieren. Das **WEP**-Workflow-Management-System wurde dazu erfolgreich bei zwei Anwendungsprojekten des DaimlerChrysler-Konzerns eingesetzt [KFW01, SAF+01]. Derzeit wird mit einem führenden Produktdatenmanagementsystemhersteller diskutiert, ob und wie sich Teile der entwickelten Konzepte in dessen Produktdatenmanagementlösung integrieren lassen.

# Kapitel 25

# Ausblick

Beschäftigt man sich intensiv mit der Lösung der in Teil I beschriebenen Praxisanforderungen, so ergeben sich fast zwangsläufig neue Problemstellungen, die im Rahmen dieser Arbeit nicht oder nicht hinreichend detailliert behandelt werden konnten und deshalb Gegenstand von Folgearbeiten sein können. Aufbauend auf der vorliegenden Arbeit bieten sich unter anderem die folgenden Themen für weitergehende Untersuchungen an:

### Zielorientierte Gruppenaktivitäten

In der bisherigen Spezifikation des **WEP**-Modells wird eine zielorientierte Aktivität durch eine *Einzelperson* bearbeitet. Es gibt jedoch Anwendungsbeispiele [SAF+01], bei denen die Bearbeitung durch eine (interdisziplinäre) Gruppe wünschenswert wäre. Auf den ersten Blick scheint eine solche Erweiterung der Spezifikation einer zielorientierten Aktivität einfach. Man legt mehrere (gleichberechtigte) Organisationselemente fest und ordnet diese den einzelnen Schrittprogrammen beziehungsweise Meilensteinspezifikationen zu.

Fragestellungen, wie die Synchronisation der potenziell parallel arbeitenden Personen, insbesondere beim Zugriff auf die verschiedenen Objektversionen im Eingangs-, Arbeits- und Ausgangsbereich sind dagegen ebenso nicht so offensichtlich lösbar wie eine praktikable Festlegung der zu informierenden Personen bei neu eingetroffenen Eingabedaten. Es muss auch untersucht werden, ob es notwendig ist, Abhängigkeiten zwischen Schrittprogrammen zu spezifizieren und, wenn ja, wie die Abhängigkeiten am sinnvollsten beschrieben werden. Spätestens bei Gruppenaktivitäten sollten auch arbeitspsychologische Fragestellungen bei der Gestaltung der komplexen Endbenutzerschnittstellen untersucht werden. Die beim **WEP**-Klienten verwendete Schreibtischmetapher [Kno99, WEP01] kann dann eventuell nicht mehr adäquat sein.

### Subscribe-Mechanismen

Erste Praxisanwendungen des **WEP**-Workflow-Management-Systems haben gezeigt, dass besonders am Sychronisationsknoten einer dynamischen Parallelität Bearbeiter auch dann vom Eintreffen aktuellerer Daten informiert werden, wenn sie sich aus Sicht des Bearbeiters in nur unwichtiger Weise geändert haben. Die Kritik des Anwenders richtet sich dabei nicht gegen das Informieren an sich – ein

konsistenter Datenfluss hat aus Anwendersicht absolute Priorität, sondern über die fehlenden Einflussmöglichkeiten, *wann* das Informieren stattzufinden hat. Die Bearbeiter wünschen sich deshalb einen benutzerfreundlichen Subscribe-Mechanismus, mit dessen Hilfe sie festlegen können, welche Änderungen ihnen sofort, zeitverzögert und kummuliert oder erst mit dem Eintreffen der endgültigen Eingabedaten gemeldet werden sollen. Ziel muss es sein, den Update-Mechanismus so zu erweitern, dass dieser einerseits die Wichtigkeit der neu eingetroffenen Daten für den Anwender berücksichtigt und anderseits nicht die Daten- und Prozesskonsistenz gefährdet.

#### Adaptive vorzeitige Datenweitergabe

In Abschnitt 20.1 wurde die Frage bereits aufgeworfen, ob der einfache ein-dimensionale **WEP**- Qualitätsstufenbegriff zur Spezifikation von SE-optimierten Datenflüssen in der Praxis ausreicht. Auch sind „intelligentere“ Algorithmen bei der Berechnung der ausführbaren Aktivitäten zu untersuchen, um bei Verzweigungen unnötige Mehrarbeit zu minimieren.

#### Zeitmanagement

Im Rahmen des **WEP**-Projekts wurde ein Algorithmus entwickelt, der bei Veränderung von Zeitvorgaben des Projektmanagements eine Neuberechnung der Meilensteinzeiten für die **WEP**-Aktivitäten durchführt [Kos00]. Aus Sicht des Projektmanagements sind auch hypothetische Anfragen wichtig, wie beispielsweise wie viele Aktivitäten ihre Meilensteinvorgaben nicht einhalten können, wenn ein assoziierter Projektmanagement-Meilenstein zeitlich vorverlegt wird. Diese Anfragen sind mit dem entwickelten Algorithmus natürlich ausführbar. Fragen einer geeigneten kummulierten Visualisierung bei einer in der Praxis üblichen großen Anzahl von Workflow-Instanzen wurden aber nicht betrachtet. Auch fehlen bisher geeignete Bewertungsfunktionen zum Vergleich von Planveränderungen unter Einbeziehungen der operativen **WEP**-Workflows und Konzepte für ein adäquates Informationsmanagement bei Meilensteinüberschreitungen.

#### Workflow-Architekturen für Entwicklungsprozesse

Die hier betrachteten Entwicklungsprozesse werden häufig weltweit und über Unternehmensgrenzen hinaus ausgeführt. Es sind zwar bereits performante und robuste Workflow-Architekturkonzepte für geographisch weit verteilte Prozesse vorgestellt worden [MAGK95, AMG+95, Bau01, JKK+02]. Ihre Anwendbarkeit auf semi-strukturierte Prozesse mit großen komplex strukturierten Datenobjekten sowie die Auswirkungen der engen Kopplung mit Produktdatenmanagementsystemen wurde bisher noch nicht hinreichend untersucht und stellen sicherlich noch ungelöste Herausforderungen dar [KRS01]. Auch sind Sicherheitsaspekte bei unternehmensübergreifenden Prozessen für die Praxis von entscheidender Bedeutung.

#### Erweiterung des Anwendungsspektrums

Die in Teil I beschriebenen Anforderungen an eine adäquate Prozessunterstützung sind nicht nur typisch für Produktentwicklungsprozesse im Speziellen, sondern auch für viele weitere Anwendungsgebiete mit semi-strukturierten Prozessen. Hierunter fallen Anwendungen aus dem medizinisch-technischen Bereich, der Software-Entwicklung, Autorensysteme sowie cross-organisatorische Prozesse aus dem E-Business-Bereich. Im letzteren Anwendungsgebiet denke man insbesondere

an die Einbindung von (wechselnden) Partnern und Zulieferern im B2B-Bereich. Jeder cross-organisatorische Prozess enthält Subprozesse der jeweils fremden Organisation, bei denen für die Geschäftspartner nur die (Zwischen-)Ergebnisse sichtbar sein sollen. Denn die internen Strukturen der Subprozesse repräsentieren interne Prozessabläufe, die häufig Geschäftsgeheimnisse darstellen und deshalb nicht sichtbar für die fremden Organisationen sein sollen.

Auch aufgrund ihrer Entstehungsgeschichte – die Arbeit entstand zum großen Teil parallel zu der Tätigkeit des Autors bei der DaimlerChrysler AG – wurde die Eignung des **WEP**-Ansatzes für die obigen Anwendungsgebiete nicht hinreichend untersucht.

Wegen der soeben skizzierten Gemeinsamkeiten sind solche Untersuchungen sicherlich lohnenswert, da die **WEP**-Konzepte auch für diese Anwendungsgebiete erfolgreich eingesetzt werden können.

Einige der angesprochenen Themen wurden bereits in Form von Diplomarbeiten oder in anderen wissenschaftlichen Projekten detaillierter betrachtet. Allerdings kann keine der Fragestellungen schon als gelöst bezeichnet werden. Es bleibt also noch viel zu tun bis Workflow-Management-Systeme hoffentlich so selbstverständlich zur Unterstützung der Produktentwicklung eingesetzt werden, wie heute Produktdatenmanagement- und CAx-Systeme. Es ist zu hoffen, dass die Ergebnisse der vorliegenden Arbeit einen Beitrag geleistet haben, dieses Ziel zu erreichen.

# Teil VI

# Literaturverzeichnis

# Literaturverzeichnis


[Aal97] W. v. d. Aalst: *Verification of Workflow Nets*. In: *Proc. Application and Theory of Petri Nets*, LNCS 1248, S. 407–426. Springer, 1997.

[ADH+92] R. Admomeit, W. Deiters, B. Holtkamp, F. Schülke und H. Weber: *K/2r: A Kernel for the ESF Software Factory Support Environment*. In: *Proc. 2nd Int. Conf. on Systems Integration*, S. 325–336, Morristown, NJ, USA, 1992.

[AGSC95] P. Antunes, N. Guimaraes, J. Segovai und J. Cardenosa: *Beyond Formal Processes: Augmenting Workflow with Group Interaction Techniques*. In: *Proc. Int. Conf. on Organizational Computing Systems (COOCS 95)*, San Jose, CA, USA, August 1995.

[AH87] M. M. Andreasen und L. Hein: *Integrated Product Development*. Springer, 1987.

[AMG+95] G. Alonso, C. Mohan, R. Günthör, D. Agrawal, A. E. Abbadi und M. Kamath: *Exotica/FMQM: A Persistent Message-Based Architecture for Distributed Workflow Management*. In: *Proc. IFIP WG8.1 Working Conf. on Information Systems for Decentralized Organizations*, Trondheim, Norway, August 1995.

[Amo01] D. Amor: *Die E-Business-(R)Evolution - Sonderausgabe - Das umfassende Executive-Briefing*. Galileo Press, 2001.

[ANRS92] M. Ansari, L. Ness, M. Rusinkiewicz und Sheth: *Using Flexible Transactions to Support Multi-system Telecommunication Applications*. In: *Proc. 18th Int. Conf. on Very Large Data Bases (VLDB)*, S. 65–76, Vancouver, Canada, August 1992.

[ASU85] A. V. Aho, R. Sethi und J. D. Ullman: *Compilers : Principles, Techniques, and Tools*. Addison-Wesley, 1985.

[Bau96] B. Baumgarten: *Petri-Netze – Grundlagen und Anwendungen*. Spektrum Akademischer Verlag, 2. Auflage, 1996.

[Bau01] T. Bauer: *Effiziente Realisierung unternehmensweiter Workflow-Management-Systeme*. Doktorarbeit, Universität Ulm, Februar 2001.

[BBC+98] N. Baker, A. Bazan, G. Chevenier, F. Estrella, Z. Kovacs, T. L. Flour, J.-M. L. Goff, S. Lieunard, R. McClatchey, S. Murray und J. Vialle: *An Object Model for Product and Workflow Data Management*. In: *Proc. Int. Conf. on Database and Expert Systems Applications (DEXA)*, Vienna, Austria, August 1998.

[BCN92] C. Batini, S. Ceri und S. Navathe: *Conceptual Database Design – An Entity-Relationship Approach*. Benjamin/Cummings Publishing Company, Redwood City, 1992.

[BD95] T. Beuter und P. Dadam: *Prinzipien der Replikationskontrolle in verteilten Systemen*. Ulmer Informatik-Berichte 95-11, Universität Ulm, November 1995.

[BD96a] T. Beuter und P. Dadam: *Anwendungsspezifische Anforderungen an Workflow-Management-Systeme am Beispiel der Domäne Concurrent-Engineering*. Ulmer Informatik-Berichte 96-04, Universität Ulm, Juni 1996.

[BD96b] T. Beuter und P. Dadam: *Prinzipien der Replikationskontrolle in verteilten Datenbanksystemen*. GI Informatik Forschung und Entwicklung, 11(4):203–212, November 1996.

[BDS98] T. Beuter, P. Dadam und P. Schneider: *The WEP Model: Adequate Workflow-Management for Engineering Processes*. In: U. Baake und R. Zobel (Herausgeber): *5th European Concurrent Engineering Conf. (ECEC)*, S. 94–98, Erlangen, Germany, April 1998.

[BFGG91] S. Bandinelli, A. Fuggetta, C. Ghezzi und S. Grigolli: *Process Enactment in SPADE*. In: J.-C. Derniame (Herausgeber): *Software Process Technology*, LNCS, S. 67–83. Springer, 1991.

[BK86] D. G. Bobrow und R. H. Katz: *Context Structures/Versioning: A Survey*. In: M. L. Brodie und J. Mylopoulos (Herausgeber): *On Knowledge Base Mangement Systems*, S. 453–459. Springer, 1986.

[BK95] D. Bogia und S. Kaplan: *Flexibility and Control for Dynamic Workflows in the Worlds Environment*. In: *COOCS* [COOCS95], S. 148–161.

[BKJ+99] T. Beuter, M. Kubicek, S. Jablonski, J. Neeb und C. Hahn: *Evaluierung von Workflow-Management-Systemen*. Interner DaimlerChrysler Evaluierungsbericht, in Zusammenarbeit mit der Universität Erlangen-Nürnberg, August 1999.

[BMR94] D. Barbará, S. Mehrotra und M. Rusinkiewicz: *INCAS: Managing Dynamic Workflows in Distributed Environments*. Technischer Bericht, Matsushita Information Technology Labratory, Princeton NY, USA, Mai 1994.

[Bre95] A. Bredenfeld: *Cooperative Concurrency Control for Design Environments*. In: *Proc. European Design Automation Conference*, Brighton, Great Britain, September 1995.

[BRJ98] G. Booch, J. Rumbaugh und I. Jacobson: *The Unified Modeling Language User Guide*. Addison-Wesley, 1998.

[BS98] U. M. Borghoff und J. H. Schlichter: *Rechnergestützte Gruppenarbeit. Eine Einführung in Verteilte Anwendungen*. Springer, 1998.

[BS01] P. Bertolazzi und M. Scannapieco: *Introducing Data Quality in a Cooperative Context*. In: *Proc. 8th Int. Conf. on Information Quality*, Boston, USA, 2001.

[BSH99] U. F. Baake, P. Stratil und D. E. Haussmann: *Optimization and Management of Concurrent Product Development Processes*. Concurrent Engineering: Research and Applications, 7(1):31–42, März 1999.

[BTW95] *Proc. GI-Fachtagung Datenbanksysteme in Büro, Technik und Wissenschaft (BTW)*, Dresden, Deutschland, März 1995.

[Bus98] C. Bussler: *Workflow Instance Scheduling with Project Management Tools*. In: *Proc. Workflow Management Workshop (WFM 98) in conjunction with 9th Int. DEXA Conf. and Workshop on Database and Expert Systems Applications (DEXA 98)*, Vienna, Austria, August 1998.

[CFM99] F. Casati, M. Fugini und I. Mirbel: *An Environment for Designing Exceptions in Workflows*. Information Systems, 24(3):255–273, 1999.

[CLK99] D. Chiu, Q. Li und K. Karplapalem: *A Meta Modeling Approach To Workflow Management Systems Supporting Exception Handling*. Information Systems, 24(2):159–184, 1999.

[CM95] N. Craven und D. Mahling: *Goals and Processes: A Task Basis for Projects and Workflows*. In: *COOCS* [COOCS95], S. 237–248.

[COOCS95] *Proc. ACM Conf. on Organizational Computing Systems*, Milpitas, CA, USA, August 1995.

[CoopIS95] *Proc. 3rd Int. Conf. on Cooperative Information Systems*, Vienna, Austria, Mai 1995.

[CS00] F. Casati und M.-C. Shan: *Process Automation as the Foundation for E-Business*. In: *Proc. 26th Int. Conf. on Very Large Data Bases (VLDB)*, S. 688–691, Cairo, Egypt, September 2000.

[DCSCW98] *Proc. D-CSCW 98 – Groupware und organisatorische Innovation*, Dortmund, Deutschland, September 1998.

[DEBul93] *Special Issue on Workflow and Extended Transaction Systems*. IEEE Data Engineering Bulletin, 16(2), Juni 1993.

[Dei97] W. Deiters: *Prozeßmodelle als Grundlage für ein systematisches Management von Geschäftsprozessen*. GI Informatik Forschung und Entwicklung, Themenheft: Workflow-Management, 12(2):52–60, Mai 1997.

[DEXA92] *Proc. Int. Conf. on Database and Expert Systems Applications (DEXA)*, Valencia, Spain, September 1992.

[DHL91] U. Dayal, M. Hsu und R. Ladin: *A Transactional Model for Long-Running Activities*. In: *Proc. 17th Int. Conf. on Very Large Data Bases (VLDB)*, S. 113–122, Barcelona, Spain, September 1991.

[Dij68] E. W. Dijkstra: *Cooperating Sequential Processes*. In: F. Genuys (Herausgeber): *Programming Languages*. Academic Press, London, Great Britain, 1968.

[DIN00] DIN (Herausgeber): *Qualitätsmanagementsysteme - Grundlagen und Begriffe (ISO 9000:2000)*. Deutsches Institut für Normung (DIN), 2000.

[DKR+95] P. Dadam, K. Kuhn, M. Reichert, T. Beuter und M. Nathe: *ADEPT: Ein integrierender Ansatz zur Entwicklung flexibler, zuverlässiger kooperierender Assistenzsysteme in klinischen Anwendungsumgebungen*. In: *GI* [GI95], S. 677–686.

[DMP97] B. Dellen, F. Maurer und G. Pews: *Knowledge-based Techniques to Increase the Flexibility of Workflow Management*. Data & Knowledge Engineering, 23(3):269–296, September 1997.

[Doy79] J. Doyle: *A True Maintenance System*. Artificial Intelligence, 12:231–272, 1979.

[DS02] Dassault Systemes: *ENOVIA LifeCycle Applications Version 5 Release 8 Service Pack Documentation*, 2002.

[Dud00] F. Dudenhöffer: *Think big: E-Business-Strategien der Automobil- und Zulieferindustrie*. CYbiz, 8:40–44, 2000.

[EGR91] C. A. Ellis, S. J. Gibbs und G. L. Rein: *Groupware – Some Issues and Experiences*. Communications of the ACM, 34(1):39–58, Januar 1991.

[EHDN00] A. v. Ende, M. Helmke, M. Dobermann und T. Naumann: *A Concept for an Adaptive Process Management in a Distributed Product Development Environment*. In: *ProSTEP* [ProSTEP00], S. 26–36.

[Ehr95] K. Ehrlenspiel: *Integrierte Produktentwicklung: Methoden für Prozessorganisation, Produkterstellung und Konstruktion*. Hanser, 1995.

[EKH+01] A. v. Ende, O. Kallmeyer, M. Helmke, M. Dobermann und M.-M. Schmidt: *Adaptives Prozessmanagement für verteilte Produktentstehungsprozesse*. In: F.-L. Krause, T. Tang und U. Ahle (Herausgeber): *Leitprojekt integrierte Virtuelle Produktentwicklung – Fortschrittsbericht*, S. 10–23. bmb+f, 2001.

[EL95] J. Eder und W. Liebhart: *The Workflow Activity Model WAMO*. In: *CoopIS* [CoopIS95], S. 87–98.

[Elm92] A. K. Elmagarmid (Herausgeber): *Database Transaction Models for Advanced Applications*. Morgan Kaufmann, 1992.

[EN99] R. A. Elmasri und S. B. Navathe: *Fundamentals of Database Systems*. Addison-Wesley, 1999.

[ENK+02] A. v. Ende, T. Naumann, O. Kallmeyer, M. Helmke, A. Hoffmann, T. Beuter und R. Knoll: *Adaptives Prozessmanagement für verteilte Produktentstehungsprozesse*. In: Frank-Lothar Krause et al. [KTA02], S. 33–40.

[EPW94] W. Eversheim, A. Pollack und M. Walz: *Auch Entwicklungsprozesse sind planbar: Workflow-Management unterstützt die Auftragsabwicklung*. VDI-Z, 136(6):78–83, 1994.

[EWM+98] W. Eversheim, M. Weck, W. Michaeli, M. Nagl, B. Westfechtel und O. Spaniol: *Die Integrationsproblematik und der SUKITS-Ansatz*. In: M. Nagl und B. Westfechtel [NW98], S. 3–14.

[FM01a] A. Frank und B. Mitschang: *Agent Protocols for Integration and Cooperation in a Design Application Framework*. In: *Proc. 2nd Int. Workshop on Software Agents and Workflow for Systems Interoperability*, S. 31–38, London, ON, Canada, Juli 2001.

[FM01b] A. Frank und B. Mitschang: *On Sharing of Objects in Concurrent Design*. In: *Proc. 6th Int. Conf. on CSCW in Design (CSCWID)*, S. 71–76, London, ON, Canada, Juli 2001.

[Fra97] E. Frankenberger: *Arbeitsteilige Produktentwicklung - Empirische Untersuchung und Empfehlungen zur Gruppenarbeit in der Konstruktion*, Band 291 der Reihe *Fortschrittsberichte VDI*. Verein Deutscher Ingenieure (VDI), 1997.

[Fra99] A. Frank: *Towards an Activity Model for Design Applications*. In: *Proc. 14th Int. Conf. on Computers and Their Applications*, Cancun, Mexico, April 1999.

[Fri88] G. Fricke: *Konstruieren als flexibler Problemlöseprozess: empirische Untersuchung über erfolgreiche Strategien und methodische Vorgehensweisen beim Konstruieren* , Band 227 der Reihe *Fortschrittsberichte VDI*. Verein Deutscher Ingenieure (VDI), 1988.

[FSM00] A. Frank, J. Sellentin und B. Mitschang: *TOGA – A Customizable Service for Data-centric Collaboration*. Information Systems, 25(2):157–176, April 2000.

[GCE00] J. Gausemeier, R. Czubayko und R. Eckes: *Integration of Cross-Domain Product Information by Means of Intelligent Solution Elements*. In: *ProSTEP* [ProSTEP00], S. 51–63.

[GGHV95] F. Gebhardt, E. Groß, T. Hemmannn und H. Voß: *Knowledge-Engineering mit MoMo*. KI Künstliche Intelligenz, 1:22–27, Januar 95.

[GHS95] D. Georgakopoulos, M. F. Hornick und A. Sheth: *An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure*. Distributed and Parallel Databases, 3(2):119–153, 1995.

[GI95] *Proc. GI-Jahrestagung*, Zürich, Schweiz, September 1995.

[GMGK+91] H. Garcia-Molina, D. Gawlick, J. Klein, K. Kleissner und K. Salem: *Modeling Long-Running Activities as Nested Sagas*. IEEE Data Engineering Bulletin, 14(1):14–18, März 1991.

[GMS87] H. Garcia-Molina und K. Salem: *Sagas*. In: *Proc. ACM SIGMOD Int. Conf. on Management of Data*, S. 249–259, San Francisco, CA, USA, Mai 1987.

[Gol96] S. Goldmann: *Procura: A Project Management Model of Concurrent Planning and Design*. In: *WETICE* [WETICE96], S. 177–183.

[GR93] J. Gray und A. Reuter: *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, 1993.

[Gri97] M. Grimm: *Temporale Aspekte in flexiblen Workflow-Management-Systemen*. Diplomarbeit, Universität Ulm, April 1997.

[Gro92] B. Groeger: *Die Einbeziehung der Wissensverarbeitung in den rechnergestützten Konstruktionsprozess*. Doktorarbeit, Technische Universität Berlin, April 1992.

[Gru93] V. Gruhn: *Entwicklung von Informationssystemen in der LION-Entwicklungsumgebung*. In: G. Scheschonk und W. Reisig [SR93].

[GS00] M. Goltz und R. Schmitt: *Product Data Controlled Engineering Workflow in the Supply Chain*. In: *ProSTEP* [ProSTEP00], S. 74–83.

[Göt98] H. Göttler: *Graphgrammatiken in der Softwaretechnik: Theorie und Anwendungen*. Informatik-Fachberichte 178. Springer, 1998.

[Hal01] E. Haller: *Digital Planen – Digital Arbeiten, Digitales Produzieren bei DaimlerChrysler*. In: *Proc. EDM-Forum*, Stuttgart, Deutschland, Juli 2001.

[Har94] R. L. Harmon: *Das Management der neuen Fabrik*. Campus-Verlag, Frankfurt, 1994.

[HB97] J. Herbst und J. Bumiller: *Towards Engineering Process Management Systems*. In: *Proc. Concurrent Engineering Europe Conf.*, S. 109–115, 1997.

[Het00] V. Hetem: *Communication: Computer Aided Engineering in the Next Millennium*. Computer-Aided Design, 32(5-6):389–394, 2000.

[HH00] K. Hergula und T. Härder: *A Middleware Approach for Combining Heterogeneous Data Sources*. In: *Proc. 1st Int. Conf. on Web Information Systems Engineering (WISE 00)*, S. 22–29, Hong Kong, Juni 2000.

[HJKW95] P. Heimann, G. Joeris, C.-A. Krapp und B. Westfechtel: *A Programmed Graph Rewriting System for Software Process Management*. In: *Proc. Joint COMPUGRAPH/-SEMAGRAPH Workshop on Graph Rewriting and Computation (SEGRAGRA '95)*, S. 123–132, Volterra, Italy, August 1995.

[HJKW96] P. Heimann, G. Joeris, C.-A. Krapp und B. Westfechtel: *DYNAMITE: Dynamic Task Nets for Software Process Management*. In: *Proc. 18th Int. Conf. on Software Engineering (ICSE 96)*, S. 331–341, Berlin, Germany, März 1996.

[HJKW97] P. Heimann, G. Joeris, C.-A. Krapp und B. Westfechtel: *Graph-Based Software Process Management*. Int. Journal of Software Engineering and Knowledge Engineering, 7(4):431–456, 1997.

[HMU00] J. E. Hopcroft, R. Motwani und J. D. Ullman: *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 2. Auflage, 2000.

[Hoh99] W. Hohaus: *B-to-B-Commerce: Zehn vor zwölf für den Einkauf*. Diebold Management Report, 10/11:7–12, 1999.

[HOR96] A. H. M. t. Hofstede, M. E. Orlowska und J. Rajapakse: *Verification Problems in Conceptual Workflow Specifications*. In: *Proc. Int. Conf. on Conceptual Modeling*, S. 73–88, Cottbus, Germany, Oktober 1996.

[HOV93] M. Hsu, R. Obermarck und R. Vuurboom: *Workflow Model and Execution*. In: *DEBul* [DEBul93], S. 45–48.

[HW98a] P. Heimann und B. Westfechtel: *Ein Workflowsystem für die Entwicklerkooperation: Außenfunktionalität und Realisierung*. In: M. Nagl und B. Westfechtel [NW98], S. 147–163.

[HW98b] P. Heimann und B. Westfechtel: *Modelle für verallgemeinerte Workflowsysteme zur Handhabung der Dynamik von Entwicklungsprozessen*. In: M. Nagl und B. Westfechtel [NW98], S. 117–145.

[IABG98a] IABG: *ProMInanD, Version 2.4 – Handbuch zum Laufzeitsystem*, 1998.

[IABG98b] IABG: *ProMInanD, Version 2.5 – Handbuch zu den Administrationswerkzeugen*, 1998.

[IBM99a] IBM: *ENOVIA Digital Enterprise Solutions, ENOVIA Portfolio, ENOVIA VPM, User's Guide, Version 1.2*, 1999.

[IBM99b] IBM: *IBM MQSeries Workflow: Getting Started with Runtime, Version 3.2.1*, 1999.

[IBM01a] IBM: *IBM MQSeries Workflow: Concepts and Architecture, Version 3.3.2*, 2001.

[IBM01b] IBM: *IBM MQSeries Workflow: Getting Started with Buildtime, Version 3.3.2*, 2001.

[IBM01c] IBM: *Lotus Solutions for the Enterprise, Volume 4. Lotus Notes and the MQSeries Enterprise Integrator*, 2001. IBM form number SG24-2217, Lotus part number 12992.

[ISO99a] ISO (Herausgeber): *Product Data Representation and Exchange: Application Protocol: Core Data For Automotive Design Processes*. ISO/DIS 10303-214. International Organization for Standardization (ISO), 1999.

[ISO99b] ISO (Herausgeber): *Product Data Representation and Exchange: Application Protocol: Electrotechnical Design And Installation*. ISO/DIS 10303-212. International Organization for Standardization (ISO), 1999.

[Jab94] S. Jablonski: *MOBILE: A Modular Workflow Model and Architecture*. In: *Proc. 4th. Int.Ẇorking Conf. on Dynamic Modelling and Information Systems*, Noordwijkerhout, The Netherlands, September 1994.

[Jab95] S. Jablonski: *Workflow-Management-Systeme, Modellierung und Architektur*. Thomson, 1995.

[Jae96] P. Jaeschke: *Geschäftsprozeßmodellierung mit INCOME*. In: G. Vossen und J. Becker [VB96], S. 141–163.

[JB96] S. Jablonski und C. Bussler: *Workflow Management - Modeling Concepts, Architecture and Implementation*. Thompson, 1996.

[JBS97] S. Jablonski, M. Böhm und W. Schulze: *Workflow-Management: Entwicklung von Anwendungen und Systemen; Facetten einer neuen Technologie*, Kapitel 18: Workflow, Staffware und InConcert im praktischen Vergleich, S. 435.440. dpunkt-Verlag, 1997.

[JH95] S. Jacobs und R. Holten: *Goal Driven Business Modelling: Supporting Decision Making within Information Systems Development*. In: *Proc. Conf. on Organizational Computing Systems*, S. 96–105, 1995.

[JKK+02] J. Jordan, U. Kleinhans, O. Kulendik, J. Porscha, A. Pross und R. Siebert: *Transparent and Flexible Cross-Organizational Workflows for Engineering Cooperations in Vehicle Development*. In: *Proc. Product Data Technology Europe (PDT Europe 2002)*, S. 101–108, Turin, Italy, Mai 2002.

[JKN+98] D. Jäger, C.-A. Krapp, M. Nagl, A. Schleicher und B. Westfechtel: *Anpaßbares und reaktives Administrationssystem für die Projektkoordination*. In: M. Nagl und B. Westfechtel [NW98], S. 331–348.

[Kar94] B. Karbe: *Flexible Vorgangssteuerung mit ProMInanD*. In: U. Hasenkamp, S. Kirn und M. Syring (Herausgeber): *CSCW – Computer Supported Cooperative Work: Informationssysteme für dezentralisierte Unternehmensstrukturen*, S. 117–133. Addison-Wesley, 1994.

[Kat90] R. D. Katz: *Toward a Unified Framework for Version Modeling in Engineering Databases*. ACM Computing Surveys, 22(4):375–408, Dezember 1990.

[Kes96] S. Kessler: *SIFRAME – The CONSENS Framework*. In: H.-J. Bullinger und J. Warschat (Herausgeber): *Concurrent Simultaneous Engineering Systems*, S. 117–139. Springer, 1996.

[KFW01] R. Knoll, M. Feltes und R. Winterstein: *Experiences in Modelling and Supporting Engineering Workflows in a Distributed and Cooperative Environment at Daimler-Chrysler*. In: U. Baake, J. Herbst und S. Schwarz (Herausgeber): *Proc. 8th European Concurrent Engineering Conf. (ECEC)*, S. 119–124, Valencia, Spain, April 2001.

[KGZH95] C. Kelling, R. German, A. Zimmermann und G. Hommel: *TimeNET – Ein Werkzeug zur Modellierung mit zeiterweiterten Petri-Netzen*. it + ti, Informationstechnik und Technische Informatik, 37(3):21–27, Juni 1995.

[KK94] R. Koller und N. Kastrup: *Prinziplösungen zur Konstruktion technischer Produkte*. Springer, 1994.

[KKSW98] C.-A. Krapp, S. Krüppel, A. Schleicher und B. Westfechtel: *Graph-Based Models for Managing Development Processes, Resources, and Products*. In: *Proc. 6th Int. Workshop on Theory and Application of Graph Transformation (TAGT 98)*, S. 455–474, Paderborn, Germany, November 1998.

[KM94] D. Krause und H. Meerkamm: *Von der Wirkstruktur zum Entwurf – Wissensbereitstellung durch Lösungskataloge*. In: J. Gausemeier (Herausgeber): *Proc. GI-Fachtagung CAD 94 – Produktdatenentwicklung und Prozessmodellierung als Grundlage neuer CAD-Systeme*, S. 405–419, Paderborn, Deutschland, 1994.

[KMS+95] K. Klöckner, P. Mambrey, M. Sohlenkamp, L. F. W. Prinz, S. Kolvenbach, U. Pankoke-Babatz und A. Syri: *POLITeam – Bridging the Gap between Bonn and Berlin for and with the Users*. In: *Proc. 4th European Conf. on Computer-Supported Cooperative Work (ECSCW 95)*, Stockholm, Sweden, September 1995.

[Kno99] R. Knoll: *Eine benutzerfreundliche graphische Repräsentation neuer Workflow-Interaktionsformen für Produktentwicklungsprozesse*. Diplomarbeit, Universität Ulm, August 1999.

[Kos00] P. Kosan: *Zeitmanagement in flexiblen Engineering-Workflows*. Diplomarbeit, Universität Ulm, Juli 2000.

[KR91] B. Karbe und N. Ramsperger: *Concepts and Implementation of Migrating Office Processess*. In: *Proc. Int. GI-Kongreß Wissensbasierte Systeme: Verteilte Künstliche Intelligenz und kooperatives Arbeiten*, S. 136–147, München, Germany, Oktober 1991.

[Kra96] C.-A. Krapp: *Parameterization of a Management Environment for the Development of Complex Products*. Technischer Bericht, Forschungsbeiträge des Aachener Graduiertenkollegs Informatik und Technik, 1996.

[Kra97] C.-A. Krapp: *A Process Management Environment for the Development of Complex Products*. In: *Proc. 3rd KI-Workshop*, Bremen, Germany, 1997.

[KRS98] W. Käfer, N. Ritter und H. Schöning: *Konfigurierungskonzepte für datenbank-basierte, technische Entwurfsanwendungen*. GI Informatik Forschung und Entwicklung, 13(1):1–17, 1998.

[KRS01] O. Kulendik, K. Rothermel und R. Siebert: *Cross-organizational workflow management – General Approaches and their Suitability for Engineering Processes*. In: *Proc. 1st IFIP-Conf. on E-Commerce, E-Business, E-Government*, S. 143–158, Zürich, Switzerland, Oktober 2001.

[KS95] F.-L. Krause und J. Schlingheider: *Development and Design with Knowledge-based Software Tools - An Overview*. Expert Systems with Applications, 8(2):233–248, 1995.

[KTA02] F.-L. Krause, T. Tang und U. Ahle (Herausgeber): *Leitprojekt integrierte Virtuelle Produktentwicklung – Abschlussbericht*. bmb+f, 2002.

[Kub98] M. Kubicek: *Organisatorische Aspekte in flexiblen Workflow-Management-Systemen*. Diplomarbeit, Universität Ulm, März 1998.

[KYH95] K. Karlapalem, H. P. Yeung und P. C. K. Hung: *CapBasED-AMS – A Framework for Capability-Based and Event-Driven Activity Management System*. In: *CoopIS* [CoopIS95], S. 205–219.

[LA94] F. Leymann und W. Altenhuber: *Managing Business Processes as an Information Resource*. IBM Systems Journal, 33(2):326–348, 1994.

[LC93] C. Liu und R. Conradi: *Automatic Replanning of Task Networks for Process Model Evolution in EPOS*. In: *Proc. European Software Engineering Conf. (ESEC 93)*, S. 434–450, Garmisch-Partenkirchen, Germany, 1993.

[Leu91] Y. Leu: *Composing Multidatabase Applications using Flexible Transactions*. IEEE Data Engineering Bulletin, 14(1):29–33, März 1991.

[Ley95] F. Leymann: *Supporting Business Transactions Via Partial Backward Recovery In Workflow Management Systems*. In: *BTW* [BTW95].

[Ley97] F. Leymann: *Transaktionsunterstützung für Workflows*. GI Informatik Forschung und Entwicklung, Themenheft: Workflow-Management, 12(2):82–90, Mai 1997.

[Lüh94] H. Lührsen: *ISO-Norm STEP/EXPRESS – Einsatz objektorientierter Technologie in der Produktdatenverwaltung*. In: S. Jänichen (Herausgeber): *Innovative Softwaretechnologien: Neue Wege mit objektorientierten Methoden und Client/Server-Architekturen*, S. C614.01–C614.15. 1994.

[LM88] C. Lin und D. C. Marinescu: *Stochastic High - Level Petri Nets and Applications*. IEEE Trans. on Computers, 37(7):815–825, 1988.

[Lotus99] Lotus Development Corporation: *Lotus Notes Release 5: Step by Step – A Beginner's Guide to Lotus Notes*, 1999.

[LSK98] D. Lutters, A. H. Streppel und H. J. J. Kals: *Product Information Structure as the Basis for the Control of Design and Engineering Processes*. CIRP Journal of Manufacturing Systems, 27(2):199–204, 1998.

[MAGK95] C. Mohan, G. Alonso, R. Günthör und M. Kamath: *Exotica: A Research Perspective on Workflow Management Systems*. IEEE Data Engineering Bulletin, 18(1):19–26, März 1995.

[Matrix01a] Matrix One: *eMatrix Basics, Version 9*, 2001.

[Matrix01b] Matrix One: *eMatrix Business Modeler Guide, Version 9*, 2001.

[Matrix01c] Matrix One: *eMatrix eMatrix Installation Guide, Version 9*, 2001.

[Mau96] F. Maurer: *Project Coordination in Design Processses*. In: *WETICE* [WETICE96], S. 191–197.

[MCB84] M. A. Marsan, G. Conte und G. Balbo: *A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems*. ACM Trans. on Computer Systems, 2(2):93–122, Mai 1984.

[MDB+00] F. Maurer, B. Dellen, F. Bendeck, S. Goldmann, H. Holz, B. Kötting und M. Schaaf: *Merging Project Planning and Web-Enabled Dynamic Workflow Technologies*. IEEE Internet Computing, 4(3):65–74, Mai/Juni 2000.

[MDEF01] E. Müller, P. Dadam, J. Enderle und M. Feltes: *Tuning an SQL-Based PDM System in a Worldwide Client/Server Environment*. In: *Proc. Int. Conf. on Data Engineering*, S. 99–108, Heidelberg, Germany, April 2001.

[Mey96] J. Meyer: *Anforderungen an zukünftige Workflow-Management-Systeme: Flexibilisierung, Ausnahmebehandlung und Dynamisierung – Erörterung am Beispiel medizinisch-organisatorischer Abläufe*. Diplomarbeit, Universität Ulm, April 1996.

[MHR96] B. Mitschang, T. Härder und N. Ritter: *Design Management in CONCORD: Combining Transaction Management, Workflow Management and Cooperation Control*. In: *Proc. 6th Int. Workshop on Research Issues in Data Engineering: Interoperability in Multidatabase Systems (RIDE-IMS)*, New Orleans, USA, Februar 1996.

[MLK01] R. J. Mentink, D. Lutters und H. J. J. Kals: *Information Management; Implications for Workflow Management & Knowledge Based Engineering*. In: *Proc. Int. Conf. on Competitive Manufacturing (COMA 01)*, S. 65–72, Stellenbosch, The Netherlands, Januar 2001.

[MMJL95] M. Merz, K. Müller-Jones und W. Lamersdorf: *Petrinetz-basierte Modellierung und Steuerung unternehmensübergreifender Geschäftsprozesse*. In: *GI* [GI95], S. 215–222.

[MP92] Z. Manna und A. Pnueli: *The Temporal Logic of Reactive and Concurrent Systems-Specification*. Springer, 1992.

[MP95] F. Maurer und G. Pews: *Ein Knowledge-Engineering-Ansatz für kooperatives Design am Beispiel der Bebauungsplanung*. KI, 1:28–34, Januar 1995.

[MR99] R. Müller und E. Rahm: *Rule-Based Dynamic Modification of Workflows in a Medical Domain*. In: *Proc. GI-Fachtagung Datenbanksysteme in Büro, Technik und Wissenschaft (BTW)*, S. 429–448, Freiburg, Deutschland, März 1999.

[MS91] U. Mehlhaus und S. Schneider: *Die Schemabeschreibungssprache EXPRESS des STEP-Standards und technische Datenbanksysteme – Eine Analyse –*. In: *Proc. GI-Fachtagung Datenbanksysteme in Büro, Technik und Wissenschaft (BTW)*, Kaiserslautern, Deutschland, März 1991.

[MS93] D. R. McCarthy und S. K. Sarin: *Workflow and Transactions in InConcert*. In: *DEBul* [DEBul93], S. 53–56.

[NC96] M. N. Nguyen und R. Conradi: *Towards a Rigorous Approach for Managing Process Evolution*. In: *Proc. 15th Int. Conf. on Conceptual Modeling (ER 96)*, S. 18–35, Cottbus, Germany, September 1996.

[NM93] K. Neumann und M. Morlock: *Operations Research*. Hanser, 1993.

[NW98] M. Nagl und B. Westfechtel (Herausgeber): *Integration von Entwicklungssystemen in Ingenieuranwendungen: Substantielle Verbesserung der Entwicklungsprozesse*. Springer, 1998.

[Obe90] A. Oberweis: *Zeitstrukturen für Informationssysteme*. Doktorarbeit, Universität Mannheim, Juli 1990.

[Obe94a] A. Oberweis: *INCOME/STAR: Methodology and Tools for the Development of Distributed Information Systems*. Information Systems, 19(8):643–660, Dezember 1994.

[Obe94b] A. Oberweis: *Workflow Management in Software Engineering Projects*. Technischer Bericht 288, Universität Karlsruhe, Institut für angewandte Informatik und formale Beschreibungsverfahren, Januar 1994.

[Obe96] A. Oberweis: *Modellierung und Ausführung von Workflows mit Petri-Netzen*. Teubner-Reihe Wirtschaftsinformatik. Teubner, 1996.

[OMG00] OMG (Herausgeber): *Product Data Management Enablers Specification (1.3)*. Object Management Group (OMG), 2000.

[OMG01] OMG (Herausgeber): *OMG Unified Modeling Language Specification (1.4 draft)*. Object Management Group (OMG), Februar 2001.

[Orr98] K. Orr: *Data Quality and Systems Theory*. Communications of the ACM, 41(2):66–71, Februar 1998.

[OSS93] A. Oberweis, P. Sander und W. Stucky: *Petri Net Based Modelling of Procedures in Complex Object Database Applications*. In: *Proc. IEEE 17th Annual Int. Computer Software and Applications Conf. (COMPSAC 93)*, S. 138–144, Phoenix, AZ, USA, November 1993.

[OSS+97] A. Oberweis, R. Schätzle, W. Stucky, W. Weitz und G. Zimmermann: *INCOME/WF - A Petri Net Based Approach to Workflow Management*. In: H. Krallmann (Herausgeber): *Wirtschaftsinformatik 97*, S. 557–580. Springer, 1997.

[Pah99] G. Pahl: *Denk- und Handlungsweisen beim Konstruieren*. Konstruktion, 51(6):11–17, Juni 1999.

[Par97] P. Parnes: *The mStar Environment: Distributed Teamwork using IP Multicast*. Technischer Bericht, Department of Computer Science, Luleå University of Technology, Sweden, September 1997.

[Par98] H. Partsch: *Requirements-Engineering systematisch. Modellbildung für softwaregestützte Systeme*. Springer, 1998.

[PD89] P. Pistor und P. Dadam: *The Advanced Information Management Prototype*. LNCS, 361:3–26, 1989.

[Pet62] C. A. Petri: *Kommunikation mit Automaten*. Doktorarbeit, TU Darmstadt, 1962.

[Pet93] C. J. Petri: *The Redux' Server*. In: *Proc. Int. Conf. on Intelligent and Cooperative Information Systems (ICICIS)*, S. 134–143, Rotterdam, The Netherlands, 1993.

[PK88] C. Pu und G. E. Kaiser: *Split-Transactions for Open-Ended Activities*. In: *Proc. 14th Int. Conf. on Very Large Data Bases (VLDB)*, S. 26–37, Los Angeles, CA, USA, September 1988.

[Pro01] Promatis: *INCOME Process Designer: User Guide*, 2001.

[ProSTEP00] *Proc. ProSTEP Science Days 2000*, Stuttgart, Germany, September 2000.

[PS97] W. Prinz und A. Syri: *Two Complementary Tools for the Cooperation in a Ministerial Environment*. Journal of Universal Computer Science, 3(8):843–864, August 1997.

[PSS97] P. Parnes, K. Synnes und D. Schefström: *The CDT mStar Environment: Scalable Distributed Teamwork in Action*. In: *Proc. Int. ACM SIGGROUP Conf. on Supporting Group Work*, S. 167–176, Phoenix, USA, November 1997.

[PTC01a] PTC: *Windchill Customizer's Guide, Release 6.2*, 2001.

[PTC01b] PTC: *Windchill User's Guide, Release 6.2*, 2001.

[RBFD01] M. Reichert, T. Bauer, T. Fries und P. Dadam: *Realisierung flexibler, unternehmensweiter Workflow-Anwendungen mit ADEPT*. In: *Proc. Elektronische Geschäftsprozesse – Grundlagen, Sicherheitsaspekte, Realisierungen, Anwendungen*, S. 217–228, Klagenfurt, Österreich, September 2001.

[RC00] K. Rouibah und K. Caskey: *An Engineering Workflow System for the Management of Engineering Processes Across Company Borders*. In: *Advances in Concurrent Engineering - CE2000, Proc. 7th ISPE Int. Conf. on Concurrent Engineering: Research and Applications*, S. 3–12, Lyon, France, Juli 2000.

[RCS00] K. Rouibah, K. Caskey und R. Schmitt: *A Parameter Based Approach to Product Data Management within Dynamic Networks*. In: *2nd Conf. on Management and Control of Production and Logistics*, S. 375–382, Grenoble, France, Juli 2000.

[RD97] M. Reichert und P. Dadam: *A Framework for Dynamic Changes in Workflow Managment Systems*. In: *Proc. Int. Conf. on Database and Expert Systems Applications (DEXA)*, S. 42–48, Toulouse, France, September 1997.

[RD98] M. Reichert und P. Dadam: *$ADEPT_{flex}$ – Supporting Dynamic Changes of Workflows Without Loosing Control*. Journal of Intelligent Information Systems (JIIS), Special Issue on Workflow and Process Management, 10(2):93–129, März 1998.

[RD00] M. Reichert und P. Dadam: *Geschäftsprozessmodellierung und Workflow-Mangagement – Konzepte, Systeme und deren Anwendungen*. Industrie Management, Themenheft: Modellierung und Simulation, 16(2):23–27, Juni 2000.

[Rei85] W. Reisig: *Systementwurf mit Netzen*. Springer, 1985.

[Rei86] W. Reisig: *Petrinetze*. Springer, 2. Auflage, 1986.

[Rei00] M. Reichert: *Dynamische Ablaufänderungen in Workflow-Management-Systemen*. Doktorarbeit, Universität Ulm, Juli 2000.

[RH80] C. V. Ramamoorthy und G. S. Ho: *Performance Evaluation of Asynchronous Concurrent Systems Using Petri Nets*. IEEE Trans. on Software Engineering, SE-6(5):440–449, September 1980.

[RHD98] M. Reichert, C. Hensinger und P. Dadam: *Supporting Adaptive Workflows in Advanced Application Environments*. In: *Proc. EDBT-Workshop on Workflow Management Systems*, S. 100–109, Valencia, Spain, März 1998.

[Rit96] N. Ritter: *The $C^3$-Locking-Protocol – A Concurrency Control Mechanism For Design Environments*. In: *Proc. Softwaretechnik in Automatisierung und Kommunikation (STAK 96)*, S. 95–110, München, Germany, März 1996.

[RM97] N. Ritter und B. Mitschang: *Die Assistenzfunktion kooperativer Designflows – verdeutlicht am Beispiel von CONCORD*. GI Informatik Forschung und Entwicklung, Themenheft: Workflow-Management, 12(2):91–100, Mai 1997.

[RMH+94] N. Ritter, B. Mitschang, T. Härder, M. Gesmann und H. Schöning: *Capturing Design Dynamics: The CONCORD Approach*. In: *Proc. IEEE Int.Conf. on Data Engineering*, S. 440–451, Houston, TX, USA, Februar 1994.

[RMH96] N. Ritter, B. Mitschang und T. Härder: *Conflict Management in CONCORD*. In: *Proc. 6th. Int. Conf. on Data and Knowledge Bases for Manufacturing and Engineering*, S. 81–100, Tempe, AZ, USA, Oktober 1996.

[RMHN94] N. Ritter, B. Mitschang, T. Härder und U. Nink: *Unterstützung der Ablaufsteuerung in Entwurfsumgebungen durch Versionierung und Konfigurierung*. In: *Proc. Softwaretechnik in Automatisierung und Kommunikation (STAK 94)*, S. 135–159, Ilmenau, Deutschland, März 1994.

[Rol96] D. Roller: *Verifikation von Workflows in IBM FlowMark*. In: G. Vossen und J. Becker [VB96], S. 353–368.

[Ros96] T. Rose: *Vorgangsmanagementsysteme: Modellierungs- und Implementierungskonzepte*. In: G. Vossen und J. Becker [VB96], S. 319–334.

[Roz97] G. Rozenberg (Herausgeber): *Handbook of Graph Grammars and Computing by Graph Transformation. Volume 1: Foundations*. World Scientific, 1997.

[RSVW94] W. Reinhard, J. Schweitzer, G. Völksen und M. Weber: *CSCW Tools: Concepts and Architectures*. IEEE Computer, 27(5):28–36, Mai 1994.

[RW91] A. Reuter und H. Wächter: *The ConTract Model*. IEEE Data Engineering Bulletin, 14(1):39–43, März 1991.

[RW92] B. Reinwald und H. Wedekind: *Automation of Control and Data flow in Distributed Application Systems*. In: *DEXA* [DEXA92], S. 475–481.

[Sac95] P. Sachs: *Transforming Work: Collaboration, Learning and Design*. Communications of the ACM, 38(9):36–44, September 1995.

[SAF+01] J. Schwarz, N. Adameck, D. Frank, R. Siebert und S. Haasis: *The Use of Feature-Based Workflow Techniques in Automotive Product Development*. In: *Proc. 7th Int. Conf. on Concurrent Enterprising (ICE 2001)*, S. 45–52, Bremen, Germany, Juni 2001.

[Sau98] G. Sauter: *Interoperabilität von Datenbanksystemen bei struktureller Heterogenität, Architektur, Beschreibungs- und Ausführungsmodell zur Unterstützung der Integration und Migration*. Doktorarbeit, Reihe DISDBIS. Infix-Verlag, Mai 1998.

[Sau99] R. Sauter: *Realisierung neuer Workflow-Interaktionsformen für Produktentwicklungsprozesse*. Diplomarbeit, Universität Ulm, Januar 1999.

[SB96] W. Schulze und M. Böhm: *Klassifikation von Vorgangsverwaltungssystemen*. In: G. Vossen und J. Becker [VB96], S. 279–293.

[SC94] M. Sohlenkamp und G. Chwelos: *Integrating Communication, Cooperation and Awareness: The DIVA Virtual Office Environment*. In: *Proc. Int. ACM Conf. of Computer-Supported Cooperative Work*, S. 331–343, Chapel Hill, USA, Oktober 1994.

[Sch00] W. Schneider: *Realisierung einer Daten- und Versionsverwaltung für flexible Engineering-Workflows*. Diplomarbeit, Fachhochschule Ulm, Juli 2000.

[SDRC99] SDRC Corporation: *Metaphase Advanced Product Configurator (APC) User's/Administrator's Manual, Version 3.2*, November 1999.

[SHL+98] N. M. Sadeh, D. W. Hildum, T. J. Laliberty, J. McA'Nulty, D. Kjenstad und A. Tseng: *A Blackboard Architecture for Integrating Process Planning and Production Scheduling*. Concurrent Engineering: Research and Applications, 6(2):88–100, Juni 1998.

[Sie97] R. Siebert: *Adaptive Workflows im Verbundprojekt PoliFlow*. In: *Proc. Workflow-Management in Geschäftsprozessen im Trend 2000*, S. 70–81, Schmalkalden, Deutschland, Oktober 1997.

[SNI95] Siemens Nixdorf Informationssysteme AG: *WorkParty V2.0, Produktinformationen*, Oktober 1995.

[Sof01] D. Sofranec: *A New Way to Do Business*. Computer aided engineering, S. 41–44, April 2001.

[SOZ95] W. Stucky, A. Oberweis und G. Zimmermann: *INCOME/STAR – Rechnergestützte Wartungs- und Entwicklungsumgebung für verteilte betriebliche Informationssysteme (Abschlussbericht)*. Universität Karlsruhe, Institut AIFB, 1995.

[SPF98] M. Sohlenkamp, W. Prinz und L. Fuchs: *POlIAwaC – Design und Evaluation des PoliTeam Awareness-Client*. In: *DCSCW* [DCSCW98], S. 181–194.

[SR93] G. Scheschonk und W. Reisig (Herausgeber): *Petri-Netze im Einsatz für Entwurf und Entwicklung von Informationssystemen*. Springer, 1993.

[Sta02a] Staffware plc: *Staffware Process Suite, Defining Staffware Procedures, Version 9, Issue 2*, Mai 2002.

[Sta02b] Staffware plc: *Staffware Process Suite, Understanding the Staffware iPE Architecture, Version 9, Issue 2*, Mai 2002.

[Sta02c] Staffware plc: *Staffware Process Suite, Using the Staffware Process Client, Version 9, Issue 2*, Mai 2002.

[SWZ95] A. Schürr, A. Winter und A. Zündorf: *Graph Grammar Engineering with PROGRES*. In: *Proc. European Software Engineering Conf. (ESEC 95)*, LNCS 989, S. 219–234. Springer, September 1995.

[Tee96] G. Teege: *HieraStates: Supporting Workflows which Include Schematic and Ad-hoc Aspects*. In: *Proc. 1st Int. Conf. on Practical Aspects of Knowledge Management (PAKM 96)*, 1996.

[TIBCO01] TIBCO: *TIBCO's Business Process Management Solutions*, 2001.

[TSMB95] S. Teufel, C. Sauter, T. Mühlherr und K. Bauknecht: *Computerunterstützung für die Gruppenarbeit*. Addison-Wesley, 1995.

[VB96] G. Vossen und J. Becker (Herausgeber): *Geschäftsprozeßmodellierung und Workflow-Management*. Thomson, 1996.

[VDI86] VDI (Herausgeber): *VDI-Richtlinie 2221: Methodik zum Entwickeln und Konstruieren technischer Systeme und Produkte*. VDI-Verlag, Düsseldorf, 1986.

[VDI96] VDI (Herausgeber): *VDI-Richtlinie 5550: System zur Zukunftssicherung - Total Quality Management (TQM)*. VDI-Verlag, Düsseldorf, 1996.

[VE92] P. Vogel und R. Erfle: *Backtracking Office Procedure*. In: *DEXA* [DEXA92], S. 506–511.

[VFS97] S. Vajna, D. Freisleben und M. Scheibler: *Vorgehensmodell als Grundlage eines effizienten Engineering*. In: R. Kasper, U. Gabbert, K.-H. Grote und S. Vajna (Herausgeber): *Entwicklungsmethoden und Entwicklungsprozesse im Maschinenbau*, S. 123–130. Logos, Berlin, Deutschland, 1997.

[VN88] N. Viswanadham und Y. Narahari: *Stochastic Petri Net Models for Performance Evaluation of Automated Manufacturing Systems*. Information and Decision Technologies, 14:125–142, 1988.

[VW00] S. Vajna und C. Weber: *Sequenzarme Konstruktion mit Teilmodellen – Ein Beitrag zur Evolution des Konstruktionsprozesses*. Konstruktion, 52(5):35–38, 2000.

[WDM+95] D. Wodtke, A. K. Dittrich, P. Muth, M. Sinnwell und G. Weikum: *Mentor: Entwurf einer Workflow-Management-Umgebung basierend auf State- und Activitycharts*. In: *BTW* [BTW95], S. 71–90.

[WEP00] DaimlerChrysler: *The WEP XML Modelling Language, V1.0*, Juni 2000.

[WEP01] DaimlerChrysler: *WEP System: User Manual, V1.3.0*, März 2001.

[Wes96] B. Westfechtel: *A Graph-based System for Managing Configurations of Engineering Design Documents*. Int. Journal of Software Engineering and Knowledge Engineering, 6(4):549–583, Dezember 1996.

[Wes01] B. Westfechtel: *Ein graphbasiertes Managementsystem für dynamische Entwicklungsprozesse*. GI Informatik Forschung und Entwicklung, 16(3):125–144, September 2001.

[WETICE96] *Proc. WET-ICE 1996*, Stanford, CA, USA, Juni 1996.

[WfMC95] WfMC (Herausgeber): *The Workflow Reference Model*. No. WfMC-TC-1003, v1.1. Workflow Management Coalition (WfMC), Januar 1995.

[WfMC99] WfMC (Herausgeber): *Workflow Management Coalition – Terminology and Glossary*. No. WFMC-TC-1011, v3.0. Workflow Management Coalition (WfMC), Februar 1999.

[WLK00] T. C. Wijker, D. Lutters und H. J. J. Kals: *The Use of Workbenches based on Information Management*. In: *Proc. University Synergy Program: Round Table*, Haifa, Israel, 2000.

[WPH+97] M. Weber, G. Partsch, S. Höck, G. Schneider, A. Scheller-Houy und J. Schweitzer: *Integrating Synchronous Multimedia Collaboration into Workflow Management*. In: *Proc. Int. ACM SIGGROUP Conf. on Supporting Group Work*, S. 281–290, Phoenix, AZ, USA, November 1997.

[WPSH+96] M. Weber, G. Partsch, A. Scheller-Huoy, J. Schweitzer und G. Schneider: *Flexible Einbindung von Telekonferenzen in Workflowmanagement*. In: H. C. Mayr (Herausgeber): *Proc. Informatik 96 – Beherrschung von Informationssystemen*. Oldenburg, 1996.

[WR90] H. Wächter und A. Reuter: *Grundkonzepte und Realisierungsstrategien des ConTract-Modells*. GI Informatik Forschung und Entwicklung, 5:202–212, 1990.

[WSF95] R. Y. Wang, V. C. Storey und C. P. Firth: *A Framework for Analysis of Data Quality Research*. IEEE Trans. on Knowledge and Data Engineering, 7(4):623–640, August 1995.

[WW96] Y. Wand und R. Y. Wang: *Anchoring Data Quality Dimensions in Ontological Foundations*. Communications of the ACM, 39(11):86–95, November 1996.

[WWD+97] G. Weikum, D. Wodtke, A. K. Dittrich, P. Muth und J. Weißenfels: *Spezifikation, Verifikation und verteilte Ausführung von Workflows in MENTOR*. GI Informatik Forschung und Entwicklung, Themenheft: Workflow-Management, 12(2):61–71, Mai 1997.

# Teil VII

# Anhang: Definitionen und Algorithmen

# Anhang A

# Die Definition der Nachfolgerrelation $Succ_{cf}$

$$
\begin{array}{rcl}
Succ_{cf} & : & ACTIVITIES^{ext} \times \wp\,(ACTIVITIES^{ext})^{1} \\
N_i.Succ_{cf} & := & \{N_j\}
\end{array}
$$

wobei für $N_j$ gilt (vergleiche die entsprechenden Nummern in Abbildung 9.8):

1. $N_j \in ACTIVITIES$, falls: $N_i \in ACTIVITIES \wedge$

$$
\begin{array}{ll}
\exists\, w \in \mathbf{T}^{*}_{\mathbf{WEP}} \mid & ControlFlow \rightarrow^{*} w \wedge \\
& w\ IsPartialWordOf\ WF \wedge \\
& w = w_1 w_2 w_3 \wedge \\
& w_2 = N_i w_{2_1} - w_{2_2} N_j \wedge \\
& Activity \rightarrow^{*} N_i \wedge \\
& Activity \rightarrow^{*} N_j \wedge \\
& w_{2_1} \quad = \quad \overbrace{\rangle \ldots \rangle}^{l>0} \ \wedge \\
& w_{2_2} \quad = \quad \overbrace{\langle \ldots \langle}^{m>0}
\end{array}
$$

2. $N_j := {}_{N_i}\blacktriangleright$, falls: $N_i \in ACTIVITIES \wedge$

$$
\begin{array}{ll}
\exists\, w \in \mathbf{T}^{*}_{\mathbf{WEP}} \mid & ControlFlow \rightarrow^{*} w \wedge \\
& w\ IsPartialWordOf\ WF \wedge \\
& w = w_1 w_2 w_3 \wedge \\
& w_2 = N_i \blacktriangleright \ \wedge \\
& Activity \rightarrow^{*} N_i
\end{array}
$$

[1] $\wp\,(S)$ bezeichne die Potenzmenge von S.

3. $N_j := {}_{N_i}\blacksquare$ , falls: $N_i \in ACTIVITIES \wedge$
$\exists\, w \in \mathbf{T}^*_{\mathbf{WEP}} \mid \; ControlFlow \rightarrow^* w \wedge$
$w\, IsPartialWordOf\; WF \wedge$
$w = w_1 w_2 w_3 \wedge$
$w_2 = N_i\blacksquare \;\wedge$
$Activity \rightarrow^* N_i$

4. $N_j \in ACTIVITIES$, falls: $N_i = \blacktriangleleft_{N_j} \wedge$
$\exists\, w \in \mathbf{T}^*_{\mathbf{WEP}} \mid \; ControlFlow \rightarrow^* w \wedge$
$w\, IsPartialWordOf\; WF \wedge$
$w = w_1 w_2 w_3 \wedge$
$w_2 = \blacktriangleleft N_j \wedge$
$Activity \rightarrow^* N_j \wedge$
$j = i$

5. $N_j \in ACTIVITIES$, falls: $N_i = \blacktriangleright_{N_i} \wedge$
$\exists\, w \in \mathbf{T}^*_{\mathbf{WEP}} \mid \; ControlFlow \rightarrow^* w \wedge$
$w\, IsPartialWordOf\; WF \wedge$
$w = w_1 w_2 w_3 \wedge$
$w_2 = \blacktriangleright\, N_j \wedge$
$Activity \rightarrow^* N_j \wedge$
$j = i$

6$a$. $N_j \in ACTIVITIES$, falls: $N_i = {}_{N_i}\blacksquare^{tv} \wedge$
$\exists\, w \in \mathbf{T}^*_{\mathbf{WEP}} \mid \; ControlFlow \rightarrow^* w \wedge$
$w\, IsPartialWordOf\; WF \wedge$
$w = w_1 w_2 w_3 \wedge$
$w_2 = w_{2_1}, w_{2_2} N_j\, w_{2_3}$
$CFdynparBranchSpec \rightarrow^* w_2 \wedge$
$Activity \rightarrow^* N_j \wedge$
$TraverseFeature \rightarrow^* w_{2_1} \wedge$
$w_{2_2} \;=\; \overbrace{\langle \ldots \langle}^{k>0}$

6$b$. $N_j := {}_{N_k}\blacktriangleright^{\circlearrowleft}$, falls: $N_i = {}_{N_i}\blacksquare^{tv} \wedge$
$\exists\, w \in \mathbf{T}^*_{\mathbf{WEP}} \mid \; ControlFlow \rightarrow^* w \wedge$
$w\, IsPartialWordOf\; WF \wedge$
$w = w_1 w_2 w_3 \wedge$
$w_2 = w_{2_1}, w_{2_2} \blacktriangleright^{\circlearrowleft}(N_k)\, w_{2_3} \wedge$
$CFdynparBranchSpec \rightarrow^* w_2 \wedge$
$CFloop \rightarrow^* \blacktriangleright^{\circlearrowleft}(N_k)\, w_{2_3} \wedge$
$TraverseFeature \rightarrow^* w_{2_1} \wedge$
$w_{2_2} \;=\; \overbrace{\langle \ldots \langle}^{l>0}$

7. $N_j \quad := N_k, \ldots, N_l, N_m, \ldots, N_p$, falls:

$$
\begin{aligned}
N_i = {} & \blacksquare^{st}_{N_i^*} \wedge \\
\exists\, w \in \mathbf{T}^*_{\mathbf{WEP}} \mid \; & ControlFlow \rightarrow^* w \wedge \\
& w \; IsPartialWordOf \; WF \wedge \\
& w = w_1 w_2 w_3 \wedge \\
& w_2 = \{ w_{2_{k_1}} N_k \, w_{2_{k_2}}, \ldots, w_{2_{l_1}} N_l \, w_{2_{l_2}}, \\
& \qquad w_{2_{m_1}} \blacktriangleright^{\circlearrowleft}(N_m) \, w_{2_{m_2}}, \ldots, w_{2_{p_1}} \blacktriangleright^{\circlearrowleft}(N_p) \, w_{2_{p_2}} \} \wedge \\
& Activity \rightarrow^* N_k \wedge \\
& \vdots \\
& Activity \rightarrow^* N_l \wedge \\
& CFloop \rightarrow^* \blacktriangleright^{\circlearrowleft}(N_m) \, w_{2_{m_2}} \wedge \\
& \vdots \\
& CFloop \rightarrow^* \blacktriangleright^{\circlearrowleft}(N_p) \, w_{2_{p_2}} \wedge \\
& w_{2_{k_1}} = \overbrace{\langle \ldots \langle}^{q>0} \wedge \\
& \vdots \\
& w_{2_{l_1}} = \overbrace{\langle \ldots \langle}^{r>0} \wedge \\
& w_{2_{m_1}} = \overbrace{\langle \ldots \langle}^{r>0} \wedge \\
& \vdots \\
& w_{2_{p_1}} = \overbrace{\langle \ldots \langle}^{s>0}
\end{aligned}
$$

$8a.\ N_j := N_k, N_l$, falls: $N_i = {}^{\circlearrowleft}\blacktriangleleft_{N_i} \wedge$

$$
\begin{aligned}
\exists\, w \in \mathbf{T}^{*}_{\mathbf{WEP}} \mid\ & ControlFlow \rightarrow^{*} w \wedge \\
& w\ IsPartialWordOf\ WF \wedge \\
& w = w_1 w_2 w_3 w_4 \wedge \\
& (w_2 = \blacktriangleright^{\circlearrowleft}(N_x) w_{2_1} (N_i)\, {}^{\circlearrowleft}\blacktriangleleft w_{2_2} w_{2_3} \vee \\
& \ w_2 = \blacktriangleright^{\circlearrowleft}(N_x)(N_i)\, {}^{\circlearrowleft}\blacktriangleleft w_{2_2} w_{2_3}) \quad \vee \\
& \ w_2 = \blacktriangleright^{\circlearrowleft}(N_x)\, {}^{\circlearrowleft}\blacktriangleleft w_{2_2} w_{2_3} \wedge l = i) \quad \wedge \\
& w_{2_2} = ((rc_k)) \wedge \\
& w_{2_3} = ((rc_l), w_{2_{3_1}} N_l w_{2_{3_2}}) \wedge \\
& w_3 = w_{3_1} N_k \wedge \\
& CFloop \rightarrow^{*} w_2 \wedge \\
& CFexitSpec \rightarrow^{*} w_{2_2} \wedge \\
& CFoptionalLoopSpec \rightarrow^{*} w_{2_3} \wedge \\
& Activity \rightarrow^{*} N_k \wedge \\
& Activity \rightarrow^{*} N_l \wedge \\
& w_{2_{3_1}} = \overbrace{\langle \ldots \langle}^{m>0} \wedge \\
& w_{3_1} = \overbrace{\langle \ldots \langle}^{n>0} \wedge
\end{aligned}
$$

$8b.\ N_j := {}_{N_k}\blacktriangleright^{\circlearrowleft}, N_l$, falls: $N_i = {}^{\circlearrowleft}\blacktriangleleft_{N_i} \wedge$

$$
\begin{aligned}
\exists\, w \in \mathbf{T}^{*}_{\mathbf{WEP}} \mid\ & ControlFlow \rightarrow^{*} w \wedge \\
& w\ IsPartialWordOf\ WF \wedge \\
& w = w_1 w_2 w_3 w_4 \wedge \\
& (w_2 = \blacktriangleright^{\circlearrowleft}(N_x) w_{2_1} (N_i)\, {}^{\circlearrowleft}\blacktriangleleft w_{2_2} w_{2_3} \vee \\
& \ w_2 = \blacktriangleright^{\circlearrowleft}(N_x)(N_i)\, {}^{\circlearrowleft}\blacktriangleleft w_{2_2} w_{2_3}) \quad \vee \\
& \ w_2 = \blacktriangleright^{\circlearrowleft}(N_x)\, {}^{\circlearrowleft}\blacktriangleleft w_{2_2} w_{2_3} \wedge l = i) \quad \wedge \\
& w_{2_2} = ((rc_k)) \wedge \\
& w_{2_3} = ((rc_l), w_{2_{3_1}} N_l w_{2_{3_2}}) \wedge \\
& w_3 = w_{3_1} \blacktriangleright^{\circlearrowleft}(N_k) w_{3_2} \wedge \\
& CFloop \rightarrow^{*} w_2 \wedge \\
& CFexitSpec \rightarrow^{*} w_{2_2} \wedge \\
& CFoptionalLoopSpec \rightarrow^{*} w_{2_3} \wedge \\
& CFloop \rightarrow^{*} w_3 \wedge \\
& Activity \rightarrow^{*} N_l \wedge \\
& w_{2_{3_1}} = \overbrace{\langle \ldots \langle}^{m>0} \wedge \\
& w_{3_1} = \overbrace{\langle \ldots \langle}^{n>0} \wedge
\end{aligned}
$$

$8c$. ohne optionalen Block

$N_j := N_k, {}_{N_l}\blacktriangleright^{\circlearrowleft}$, falls: $N_i = {}^{\circlearrowleft}\blacktriangleleft_{N_i} \wedge$

$$
\begin{aligned}
\exists\, w \in \mathbf{T}^{*}_{\mathbf{WEP}} \mid\ & ControlFlow \rightarrow^{*} w \wedge \\
& w\ IsPartialWordOf\ WF \wedge \\
& w = w_1 w_2 w_3 w_4 \wedge \\
& (w_2 = \blacktriangleright^{\circlearrowleft}(N_l) w_{2_1} (N_i)\, {}^{\circlearrowleft}\blacktriangleleft w_{2_2} w_{2_3} \vee \\
& \ w_2 = \blacktriangleright^{\circlearrowleft}(N_l)(N_i)\, {}^{\circlearrowleft}\blacktriangleleft w_{2_2} w_{2_3}) \quad \vee \\
& \ w_2 = \blacktriangleright^{\circlearrowleft}(N_l)\, {}^{\circlearrowleft}\blacktriangleleft w_{2_2} w_{2_3} \wedge l = i) \quad \wedge \\
& w_{2_2} = ((rc_k)) \wedge \\
& w_{2_3} = ((rc_l)) \wedge \\
& w_3 = w_{3_1} w_{3_2} w_{3_3} \wedge \\
& w_{3_2} = N_k \wedge \\
& CFloop \rightarrow^{*} w_2 \wedge \\
& CFblock \rightarrow^{*} w_{2_1} \wedge \\
& CFexitSpec \rightarrow^{*} w_{2_2} \wedge \\
& CFoptionalLoopSpec \rightarrow^{*} w_{2_3} \wedge \\
& Activity \rightarrow^{*} w_{3_2} \wedge \\
& w_{3_1} = \overbrace{\langle \ldots \langle}^{m>0}
\end{aligned}
$$

$8c$. mit optionalen Block

$N_j := N_k, {}_{N_l}\blacktriangleright^{\circlearrowleft}$, falls: $N_i = {}^{\circlearrowleft}\blacktriangleleft_{N_i} \wedge$

$$
\begin{aligned}
\exists\, w \in \mathbf{T}^{*}_{\mathbf{WEP}} \mid\ & ControlFlow \rightarrow^{*} w \wedge \\
& w\ IsPartialWordOf\ WF \wedge \\
& w = w_1 w_2 w_3 w_4 \wedge \\
& (w_2 = \blacktriangleright^{\circlearrowleft}(N_x) w_{2_1} (N_i)\, {}^{\circlearrowleft}\blacktriangleleft w_{2_2} w_{2_3} \vee \\
& \ w_2 = \blacktriangleright^{\circlearrowleft}(N_x)(N_i)\, {}^{\circlearrowleft}\blacktriangleleft w_{2_2} w_{2_3}) \quad \vee \\
& \ w_2 = \blacktriangleright^{\circlearrowleft}(N_x)\, {}^{\circlearrowleft}\blacktriangleleft w_{2_2} w_{2_3} \wedge l = i) \quad \wedge \\
& w_{2_2} = ((rc_k)) \wedge \\
& w_{2_3} = ((rc_l), w_{2_{3_1}} w_{2_{3_2}}) \wedge \\
& w_{2_{3_2}} = \blacktriangleright^{\circlearrowleft}(N_l) w_{2_{3_{2_1}}} \wedge \\
& w_3 = w_{3_1} w_{3_2} w_{3_3} \wedge \\
& w_{3_2} = N_k \wedge \\
& CFloop \rightarrow^{*} w_2 \wedge \\
& CFloop \rightarrow^{*} w_{2_{3_2}} \wedge \\
& CFexitSpec \rightarrow^{*} w_{2_2} \wedge \\
& CFoptionalLoopSpec \rightarrow^{*} w_{2_3} \wedge \\
& Activity \rightarrow^{*} w_{3_2} \wedge \\
& w_{2_{3_1}} = \overbrace{\langle \ldots \langle}^{m>0}
\end{aligned}
$$

$8d$. ohne optionalen Block

$N_j := {}_{N_k}\blacktriangleright^{\circlearrowleft}, {}_{N_l}\blacktriangleright^{\circlearrowleft}$, falls: $N_i = {}^{\circlearrowleft}\blacktriangleleft_{N_i} \wedge$

$$
\begin{aligned}
\exists\, w \in \mathbf{T}^{*}_{\mathbf{WEP}} \mid\ & ControlFlow \rightarrow^{*} w \wedge \\
& w\ IsPartialWordOf\ WF \wedge \\
& w = w_1 w_2 w_3 w_4 \wedge \\
& (w_2 = \blacktriangleright^{\circlearrowleft}(N_l) w_{2_1} (N_i)\, {}^{\circlearrowleft}\blacktriangleleft w_{2_2} w_{2_3} \vee \\
& \ w_2 = \blacktriangleright^{\circlearrowleft}(N_l)(N_i)\, {}^{\circlearrowleft}\blacktriangleleft w_{2_2} w_{2_3}) \quad \vee \\
& \ w_2 = \blacktriangleright^{\circlearrowleft}(N_l)\, {}^{\circlearrowleft}\blacktriangleleft w_{2_2} w_{2_3} \wedge l = i) \quad \wedge \\
& w_{2_2} = ((rc_k)) \wedge \\
& w_{2_3} = ((rc_l)) \wedge \\
& w_3 = w_{3_1} w_{3_2} w_{3_3} \wedge \\
& w_{3_2} = \blacktriangleright^{\circlearrowleft} N_k \wedge \\
& CFloop \rightarrow^{*} w_2 \wedge \\
& CFblock \rightarrow^{*} w_{2_1} \wedge \\
& CFexitSpec \rightarrow^{*} w_{2_2} \wedge \\
& CFoptionalLoopSpec \rightarrow^{*} w_{2_3} \wedge \\
& CFloop \rightarrow^{*} w_3 \wedge \\
& w_{3_1} = \overbrace{\langle \ldots \langle}^{m>0}
\end{aligned}
$$

$8e$. mit optionalen Block

$N_j := {}_{N_k}\blacktriangleright^{\circlearrowleft}, {}_{N_l}\blacktriangleright^{\circlearrowleft}$, falls: $N_i = {}^{\circlearrowleft}\blacktriangleleft_{N_i} \wedge$

$$
\begin{aligned}
\exists\, w \in \mathbf{T}^{*}_{\mathbf{WEP}} \mid\ & ControlFlow \rightarrow^{*} w \wedge \\
& w\ IsPartialWordOf\ WF \wedge \\
& w = w_1 w_2 w_3 w_4 \wedge \\
& (w_2 = \blacktriangleright^{\circlearrowleft}(N_x) w_{2_1} (N_i)\, {}^{\circlearrowleft}\blacktriangleleft w_{2_2} w_{2_3} \vee \\
& \ w_2 = \blacktriangleright^{\circlearrowleft}(N_x)(N_i)\, {}^{\circlearrowleft}\blacktriangleleft w_{2_2} w_{2_3}) \quad \vee \\
& \ w_2 = \blacktriangleright^{\circlearrowleft}(N_x)\, {}^{\circlearrowleft}\blacktriangleleft w_{2_2} w_{2_3} \wedge l = i) \quad \wedge \\
& w_{2_2} = ((rc_k)) \wedge \\
& w_{2_3} = ((rc_l), w_{2_{3_1}} w_{2_{3_2}}) \wedge \\
& w_{2_{3_2}} = \blacktriangleright^{\circlearrowleft}(N_l) w_{2_{3_{2_1}}} \wedge \\
& w_3 = w_{3_1} w_{3_2} w_{3_3} \wedge \\
& w_{3_2} = \blacktriangleright^{\circlearrowleft} N_k \wedge \\
& CFloop \rightarrow^{*} w_2 \wedge \\
& CFloop \rightarrow^{*} w_{2_{3_2}} \wedge \\
& CFexitSpec \rightarrow^{*} w_{2_2} \wedge \\
& CFoptionalLoopSpec \rightarrow^{*} w_{2_3} \wedge \\
& CFloop \rightarrow^{*} w_3 \wedge \\
& w_{2_{3_1}} = \overbrace{\langle \ldots \langle}^{m>0}
\end{aligned}
$$

9. $N_j \quad := N_k, \ldots, N_l, {}_{N_m}\blacktriangleright^{\circlearrowleft}, \ldots, {}_{N_p}\blacktriangleright^{\circlearrowleft}$, falls:

$$
\begin{array}{ll}
N_i = {}_{N_i}\blacktriangleright^{\prec} \wedge & \\
\exists\, w \in \mathbf{T}^*_{\mathbf{WEP}} \mid & ControlFlow \rightarrow^* w \wedge \\
& w\, IsPartialWordOf\ WF \wedge \\
& w = w_1 w_2 w_3 \wedge \\
& w_2 = \{(rc_k, w_{2_{k_1}} N_k\, w_{2_{k_2}}), \ldots, (rc_l, w_{2_{l_1}} N_l\, w_{2_{l_2}}), \\
& \qquad (rc_m, w_{2_{m_1}} \blacktriangleright^{\circlearrowleft}(N_m)\, w_{2_{m_2}}), \ldots, (rc_n, w_{2_{p_1}} \blacktriangleright^{\circlearrowleft}(N_p)\, w_{2_{p_2}})\} \wedge \\
& Areturncode \rightarrow^* rc_k \wedge \\
& \quad \vdots \\
& Areturncode \rightarrow^* rc_l \wedge \\
& Areturncode \rightarrow^* rc_m \wedge \\
& \quad \vdots \\
& Areturncode \rightarrow^* rc_n \wedge \\
& Activity \rightarrow^* N_k \wedge \\
& \quad \vdots \\
& Activity \rightarrow^* N_l \wedge \\
& CFloop \rightarrow^* \blacktriangleright^{\circlearrowleft}(N_m)\, w_{2_{m_2}} \wedge \\
& \quad \vdots \\
& CFloop \rightarrow^* \blacktriangleright^{\circlearrowleft}(N_p)\, w_{2_{p_2}} \wedge \\
& w_{2_{k_1}} = \overbrace{\langle \ldots \langle}^{q>0} \wedge \\
& \quad \vdots \\
& w_{2_{l_1}} = \overbrace{\langle \ldots \langle}^{r>0} \wedge \\
& w_{2_{m_1}} = \overbrace{\langle \ldots \langle}^{s>0} \wedge \\
& \quad \vdots \\
& w_{2_{p_1}} = \overbrace{\langle \ldots \langle}^{t>0}
\end{array}
$$

# Anhang B

# Algorithmen

## B.1 Der WEP-Workflow-Laufzeitgraph

Das Ziel des hier beschriebenen **WEP**-Workflow-Laufzeitgraphen ist die effiziente Umsetzung der vorzeitigen Datenweitergabe. Es werden dazu nach der Workflow-Modellierung verschiedene Hilfsstrukturen aufgebaut, die eine schnellere Auswertung zur Laufzeit ermöglichen.

### B.1.1 Aufbau des Parallelitätsgraphen $parGraph$

Der nun im Folgenden skizzierte Algorithmus *CreateParGraphCF* zum Aufbau des Paralleliätsgraphen für eine Aktivität geht zweistufig vor. Zuerst wird durch Analyse des Kontrollflusses bestimmt, welche Aktivitäten zu einer betrachteten Aktivität parallel verlaufen. Dieser Graph $parGraph^{cf}$ ist für alle Aktivitäten eines parallelen Zweigs gleich. Er wird deshalb beim Verzweigungsknoten der äußersten parallelen Verzweigung (Schachtelungstiefe 1) für jeden parallelen Zweig gespeichert. Ein Beispiel ist in Abbildung 15.6 zu sehen. Im Detail ist dieser Algorithmus in Abschnitt B.1.1.2 beschrieben.

In der zweiten Stufe – beschrieben in Abschnitt B.1.1.4 – wird danach für jede Aktivität $A$ ihr Datenfluss betrachtet und der $parGraph^{cf}$ um Aktivitäten „ausgedünnt“, die nicht mit der Aktivität $A$ in einer Schreib-Lese-Beziehung[1] stehen. Damit entsteht natürlich für jede Aktivität $A$ und jedem ihrer Ausgabeparameter $op$ sowie für jede seiner bereitgestellten Qualitätsstufen $oql$ ein eigener Parallelitätsgraph $parGraph(A, op, oql)$. Alle Parallelitätsgraphen einer Aktivität werden bei dem entsprechenden Aktivitätsknoten im **WEP**-$workflow_{ROV}$-Graph gespeichert.

#### B.1.1.1 Verwendete Datenstrukturen

Der Algorithmus *CreateParGraphCF* geht davon aus, dass beim Workflow-Graph **WEP**-$workflow_{ROV}$ die folgenden Datenstrukturen vorhanden sind:

[1] kein Datenflussweg von $A$ zu diesen Akivitäten

- Jeder parallele Verzweigungspunkt besitzt eine $parGraph^{cf}$-Tabelle mit Einträgen der Form *(firstActivityOfParBranch, parGraphCF)*. Im Algorithmus *CreateParGraphCF* wird auf eine Zeile in dieser Tabelle über den Parameter *pgte (parGraphTableEntry)* zugegriffen.
- Bei jeder Aktivität *A* wird ein Tupel *parGraphRoot* verwaltet. Das Tupel enthält die Elemente *nodeIdOfParBegin* und *firstActOfParBranch*. In *nodeIdOfParBegin* wird der parallele Verzweigungsknoten vermerkt, in dessen $parGraph^{cf}$-Tabelle in der Zeile mit *firstActOfParBranch* der für diese Aktivität *A* gültige Parallelitätsgraph $parGraph^{cf}$ gespeichert ist. Die Tupel *parGraphRoot* werden vor Start des Algorithmuses mit *(0,0)* initialisiert. Sie besitzen also noch keinen Parallelitätsgraph.

#### B.1.1.2 Der Algorithmus *CreateParGraphCF*

Der rekursiv aufgebaute Algorithmus *CreateParGraphCF* erhält als Eingabeparameter den jeweiligen Startknoten des zu untersuchenden Workflow-Graphen, einen Parallelitätszähler *pc*, der die aktuelle Schachtelungstiefe der Parallelitätskonstrukte zählt und den Parameter *pgte*, der auf die Position des zu verwendenden $parGraph^{cf}$ verweist.

Aufgerufen wird der Algorithmus mit dem Startknoten des Workflows, einem mit *0* initialisierten Parallelitätszähler *pc* und mit dem Tupel *(0,0)* als (noch nicht verhandenen) Verweis auf den aktuellen Parallelitätsgraph $parGraph^{cf}$. Der Algorithmus durchläuft dann rekursiv alle Pfade durch den Workflow-Graph, baut den $parGraph^{cf}$ auf und vermerkt für jede Aktivität, wo der für sie gültige Parallelitätsgraph gespeichert wird. Er terminiert mit der Endeaktivität des Workflows.

*CreateParGraphCF(n: $ACTIVITIES^{ext}$, pc: parCounter, pgte: parGraphTableEntry): $ACTIVITIES^{ext}$*
***begin***
*ParGraphCF := ∅;*
***switch*** *n:*
***case*** *n == ■ :*
*pc++;*
*SuccSet := n.Succ;*
*ParGraphCF := n;*
*ParGraphCF.Succ := ∅;*
*while (SuccSet ≠ ∅)* ***do***
*// faosp = first activity of selected path*
*faosp := SuccSet.GetActivity;*
***if*** *(pc == 1)* ***then***
*// Wir betreten gerade eine äußerste Parallelität. An diesem Knoten werden die parGraphen angehängt.*
***for*** *(a ∈ (SuccSet \ faosp))* ***do***
*ParGraphCF.Succ := ParGraphCF.Succ ∪ CreateParGraphCF(a, pc, (n, a));*
*// Vermerke dabei im Parameter pgte die jeweils erste Aktivität des jeweiligen Zweigs*
***endfor****;*
*// Speichere diesen parGraph an dieser äußeren Parallelität für alle Aktivitäten des Zweiges,*
*// der mit fasop beginnt*
*n.ParGraphSet:= n.ParGraphSet ∪ (fasop, ParGraphCF);*
***else***
*// Wir befinden uns in einer verschachtelten Parallelität*
***for*** *(a ∈ (SuccSet \ faosp))* ***do***
*ParGraphCF.Succ := ParGraphCF.Succ ∪ CreateParGraphCF(a, pc, pgte));*
*// Vermerke an der Aktivität, welcher parGraph für sie gilt*
*a.ParGraphRoot := pgte;*
***endfor****;*
***endif****;*

```
        SuccSet.RemoveActivity(faosp);
      endwhile;
    case n == ┣ :
      pc−−;
      ParGraphCF := n;
      if (pc > 0) then
        // Es wurde noch nicht die äußerste Parallelität beendet.
        ParGraphCF.Succ := ParGraphCF.Succ ∪ CreateParGraphCF(n.Succ, pc, pgte));
      endif;
    case n == ≻◀ :
      if (pc > 0) then
        // Es wurde noch nicht die äußerste Parallelität beendet.
        ParGraphCF := n;
        ParGraphCF.Succ := CreateParGraphCF(n.Succ, pc, pgte));
      endif;
    case n == ▶◌ :
      if (pc > 0) then
        // Es wurde noch nicht die äußerste Parallelität beendet.
        if (n.AlreadyVisited == true) then
          // Diese Schleife wurde bereits abgearbeitet.
          ParGraphCF.Succ := ∅;
        else
          ParGraphCF := n;
          ParGraphCF.Succ := CreateParGraphCF(n.Succ, pc, pgte));
          // Markiere diese Schleife als schon einmal betreten.
          n.AlreadyVisited := true;
        endif;
      endif;
    case n ∈ A:
      if (pc > 0) then
        // Es wurde noch nicht die äußerste Parallelität beendet.
        ParGraphCF := n;
      endif;
      // Vermerke an der Aktivität, welcher parGraph für sie gilt.
      n.ParGraphRoot := pgte;
      ParGraphCF.Succ := CreateParGraphCF(n.Succ, pc, pgte));
    case n == ▶≺ or n == ◌◀ :
      if (pc > 0) then
        // Es wurde noch nicht die äußerste Parallelität beendet.
        for (a ∈ n.Succ) do
          ParGraphCF.Succ :=ParGraphCF.Succ ∪ CreateParGraphCF(a, pc, pgte));
        endfor;
      endif;
    endswitch;
  return ParGraphCF;
end;
```

### B.1.1.3 Korrektheit des Algorithmus

Der Algorithmus terminiert aus zwei Gründen:

1. In jedem der Fälle der ***case***-Anweisung wird der Algorithmus rekursiv mit allen Nachfolgeaktivitäten aufgerufen, so dass man sich im Workflow-Graph immer mehr der Endeaktivität nähert.

2. Bereits besuchte Knoten werden markiert (*AlreadyVisited*), so dass Schleifen erkannt werden und nicht erneut zu rekursiven Aufrufen führen.

Es werden über die Nachfolgerrelation alle Knoten des Graphen besucht (Tiefensuche). Mittels des Zählers *pc* wird auch nur dann ein Parallelitätsgraph aufgebaut, wenn sich die betrachtete Aktivität innerhalb einer Parallelität befindet. In diesen Parallelitätsgraph befinden sich dann nur Aktivitäten aus den anderen parallelen Zweigen.

#### B.1.1.4 Einbeziehung der Datenflussabhängigkeiten

Der bisher entstandene Parallelitätsgraph berücksichtigt nur die Kontrollflussabhängigkeiten. Er kann damit für eine Aktivität $A$ auch Aktivitätsknoten $Ai$ enthalten, mit denen $A$ keine Schreib-Leseabhängigkeiten besitzt. Fehlt eine solche Datenflussabhängigkeit mit $Ai$, so kann $Ai$ aus dem Parallelitätsgraph von $A$ entfernt werden, da sicherlich $Ai$ nicht benachrichtigt werden muss, wenn $A$ Objektversionen weitergibt.

Darüber hinaus muss unterschieden werden, ob eine Aktivität nur lesend oder auch schreibend auf das Objekt zugreift. Eine nur lesende Aktivität wird bezüglich dieses Objekts nie eine $ReleaseObjectVersion$-Operation ausführen und somit auch einen Datenflusszyklus zu der betrachteten Aktivität $A$ verursachen können.

Der Umbau des Parallelitätsgraphen erfolgt in den folgenden Schritten:

1. Alle Aktivitäten, die nicht in das betrachtete Objekt mit der gerade untersuchten Qualitätsstufe schreiben, werden als *Dummy-Aktivitäten* markiert. Dabei wird vermerkt, ob sie vom Objekt in der untersuchten Qualitätsstufe lesen.

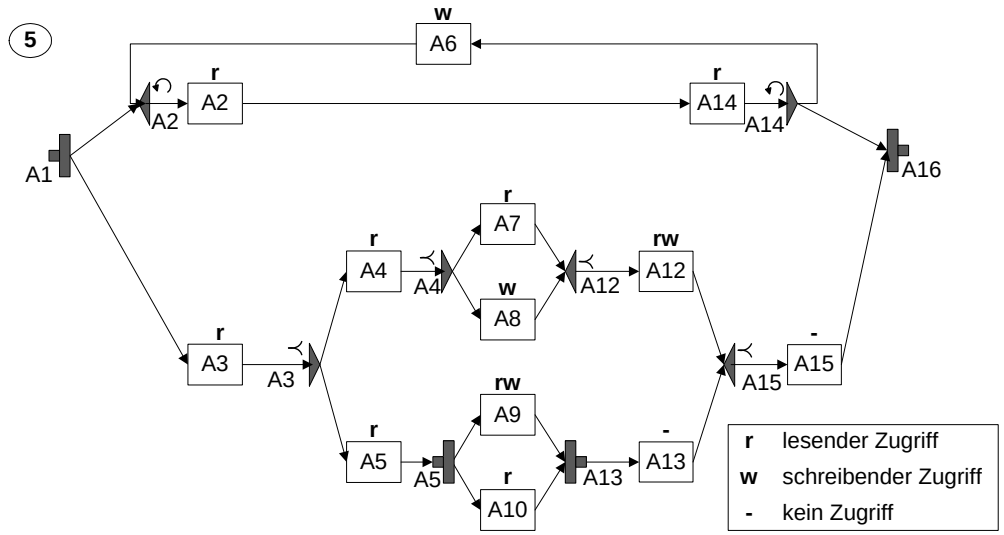
2. Für jeden Pfad durch den Parallelitätsgraph werden alle der lesenden Dummy-Aktivitäten bei der nächsten schreibenden Aktivität $A_w$ in der Menge $A_w.ROAS$ (*Read-Only Activity Set*) vermerkt. Wird $A_w$ über eine neue Objektversion informiert, so müssen auch alle Aktivitäten aus $A_w.ROAS$ benachrichtigt werden, um dem Korrektheitsbegriff der SE-Serialisierbarkeit zu genügen. Gibt es einen Pfad durch den Parallelitätsgraph, der keine schreibenden Aktivitäten enthält, so werden alle lesenden Aktivitäten beim letzten Knoten des Parallelitätsgraphen (äußerster ▸ -Knoten) in dessen $ROAS$-Menge gespeichert. Alle anderen ▸ -Knoten besitzen eine leere $ROAS$-Menge.

3. Alle Dummy-Aktivitäten, die nicht die Struktur des Parallelitätsgraphen verletzen, können gelöscht werden. Zur Strukturerhaltung bleibt also immer ein Aktivitätenknoten zwischen zwei Kontrollflussknoten erhalten. Enthält ein Pfad zwischen zwei Kontrollflussknoten nur Dummy-Aktivitäten, so muss eine Dummy-Aktivität erhalten bleiben.

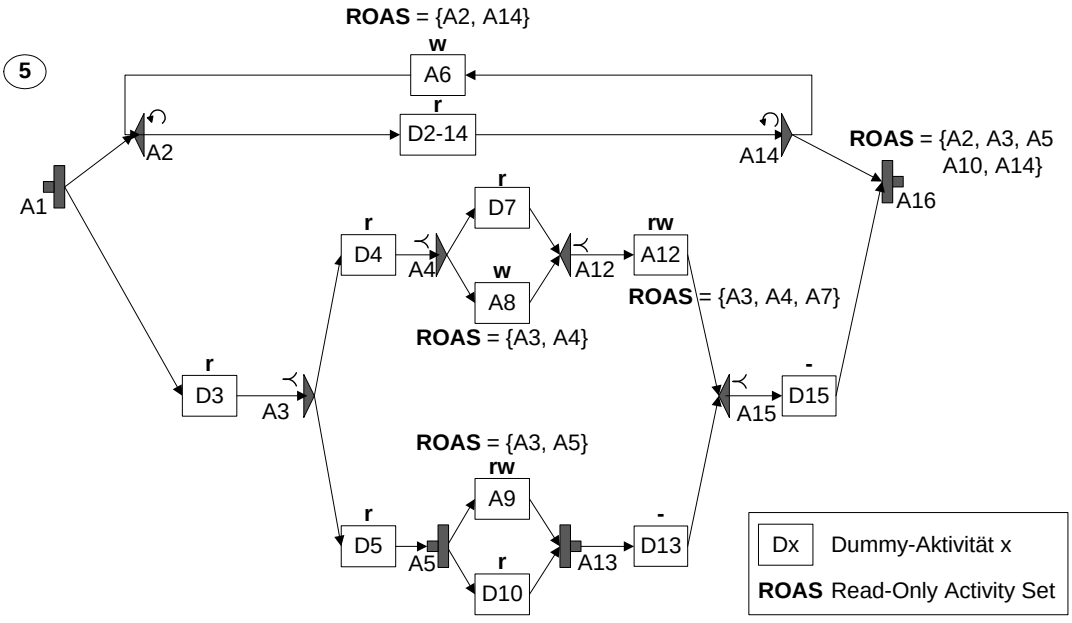
Die Abbildung B.1 zeigt exemplarisch die Transformationsschritte für den Parallelitätsgraph ⑤ aus der Abbildung 15.7.

### B.1.2 Bestimmung der potenziellen Aktivierungsmenge $potActSet^{df}_{ROV}$

Die Festlegung der potenziellen Aktivierungsmenge $potActSet^{df}_{ROV}(A, op, oql)$ für eine Aktivität $A$ und einem ihrer Ausgabeparameter $op$ mit der Qualitätsstufe $oql$ gestaltet sich weitaus einfacher.




(a) Ausgangssituation: Abweichend von der Annahme in der Abbildung 15.7, bei der alle Aktivitäten des Workflows *lesend* und *schreibend* auf das Objekt $O$ zugreifen, wird hier die Zugriffsart der Aktivitäten explizit angegegeben.




(b) Transformationsergebnis: Dummy-Aktivitäten und ROAS-Mengen bestimmt

**Abbildung B.1:** Transformationsschritte beim Parallelitätsgraph 5 aus Abbildung 15.7

Ausgehend von der betrachteten Aktivität $A$ und Ausgabeparameter $op$ wird für jede mögliche Qualitätsstufe des Parameters solange rekursiv die Nachfolgerrelation $Succ_{cf}$ ausgewertet, bis für jeden Nachfolgerpfad die erste Aktivität gefunden wurde, die über Datenflusseingangskanten mit dem Ausgabeparameter $op$ verbunden ist und die die bereitgestellte Qualitätsstufe $oql$ benötigt oder bis das Ende des Pfads erreicht wurde. Die Menge $potActSet^{df}_{ROV}(A, op, oql)$ besteht damit aus Tupeln der Form (Qualitätsstufe $oql$, Menge von Nachfolgeraktivitäten, die Eingabeparameter in Qualitätsstufe $oql$ benötigen).

## B.2 UML-Aktivitätsdiagramme des formalen WEP-Ausführungsmodells

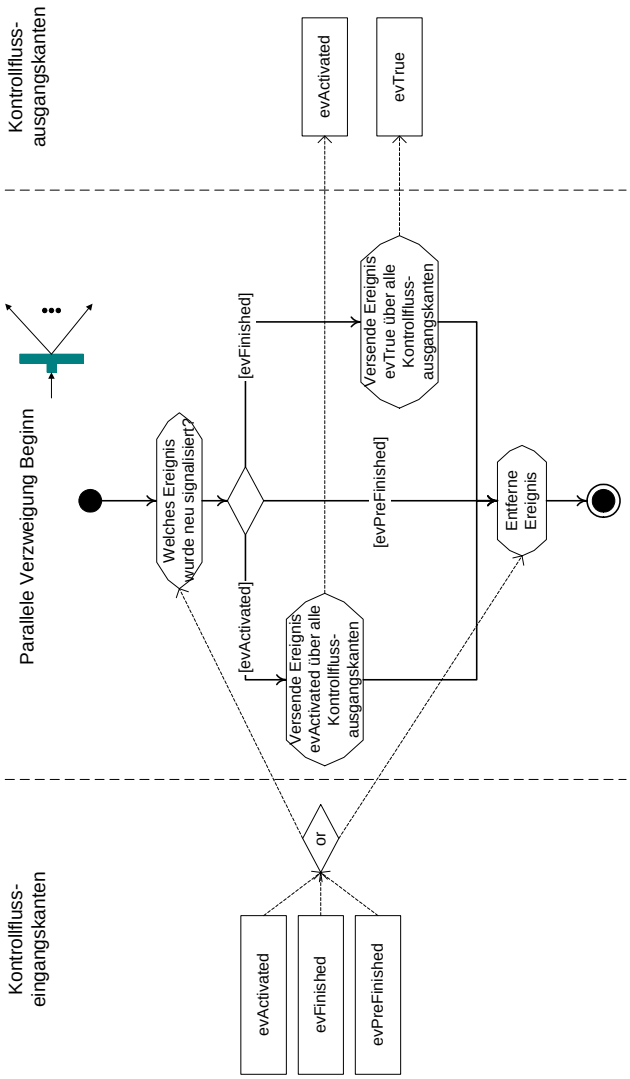



**Abbildung B.2:** Schaltlogik des Kontrollflussknotens Parallele Verzweigung Beginn

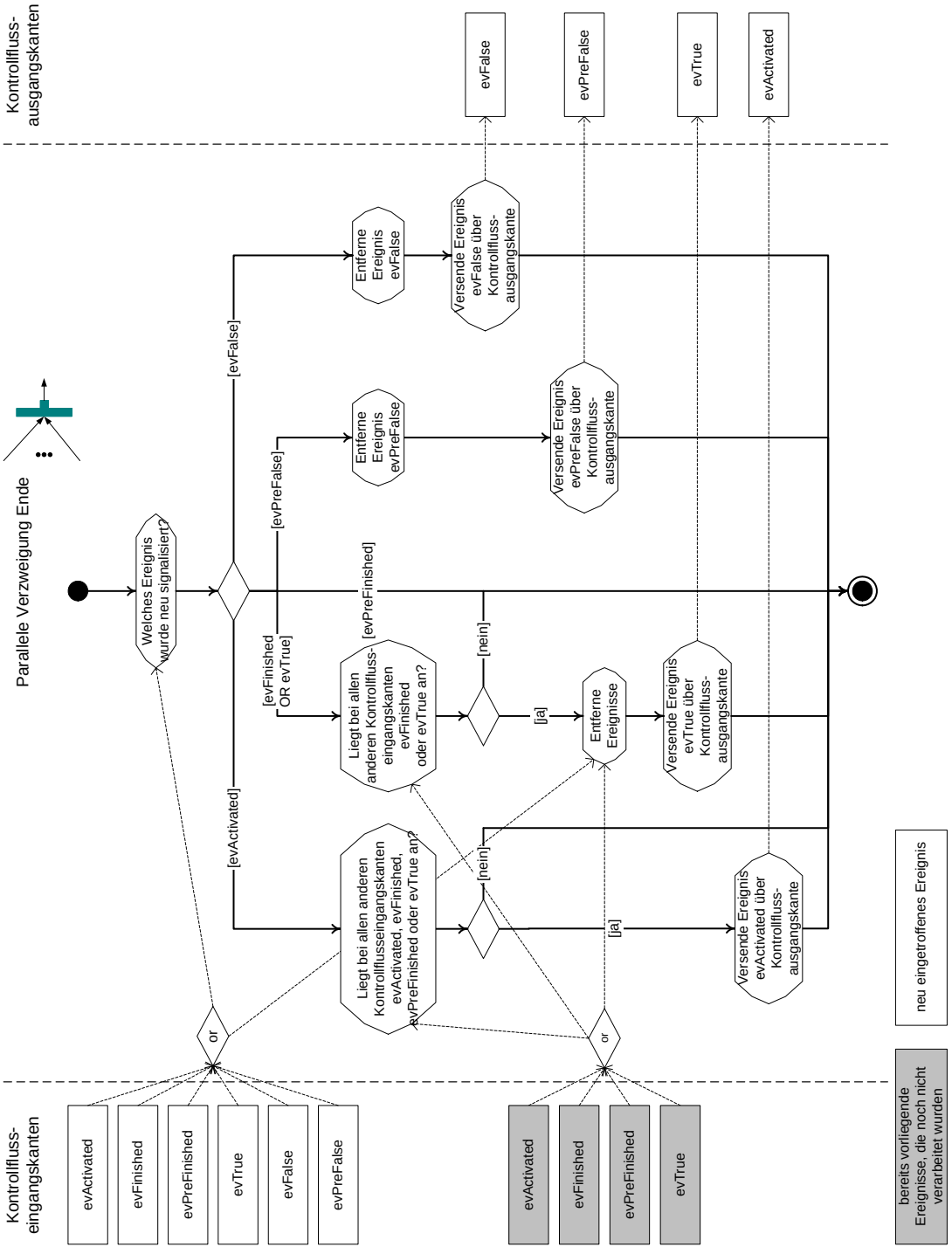



**Abbildung B.3:** Schaltlogik des Kontrollflussknotens Parallele Verzweigung Ende

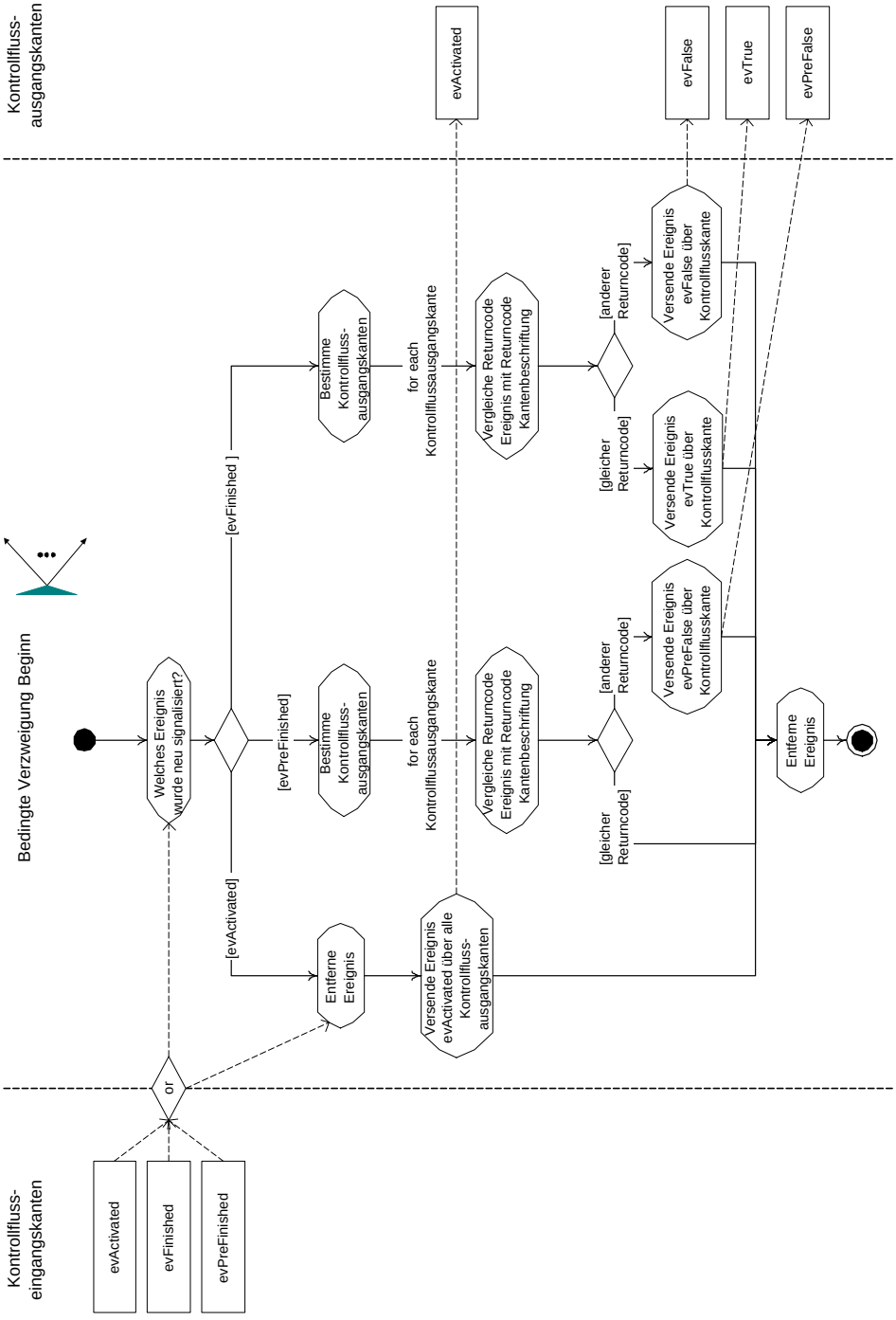



**Abbildung B.4:** Schaltlogik des Kontrollflussknotens Verzweigung Beginn ▶

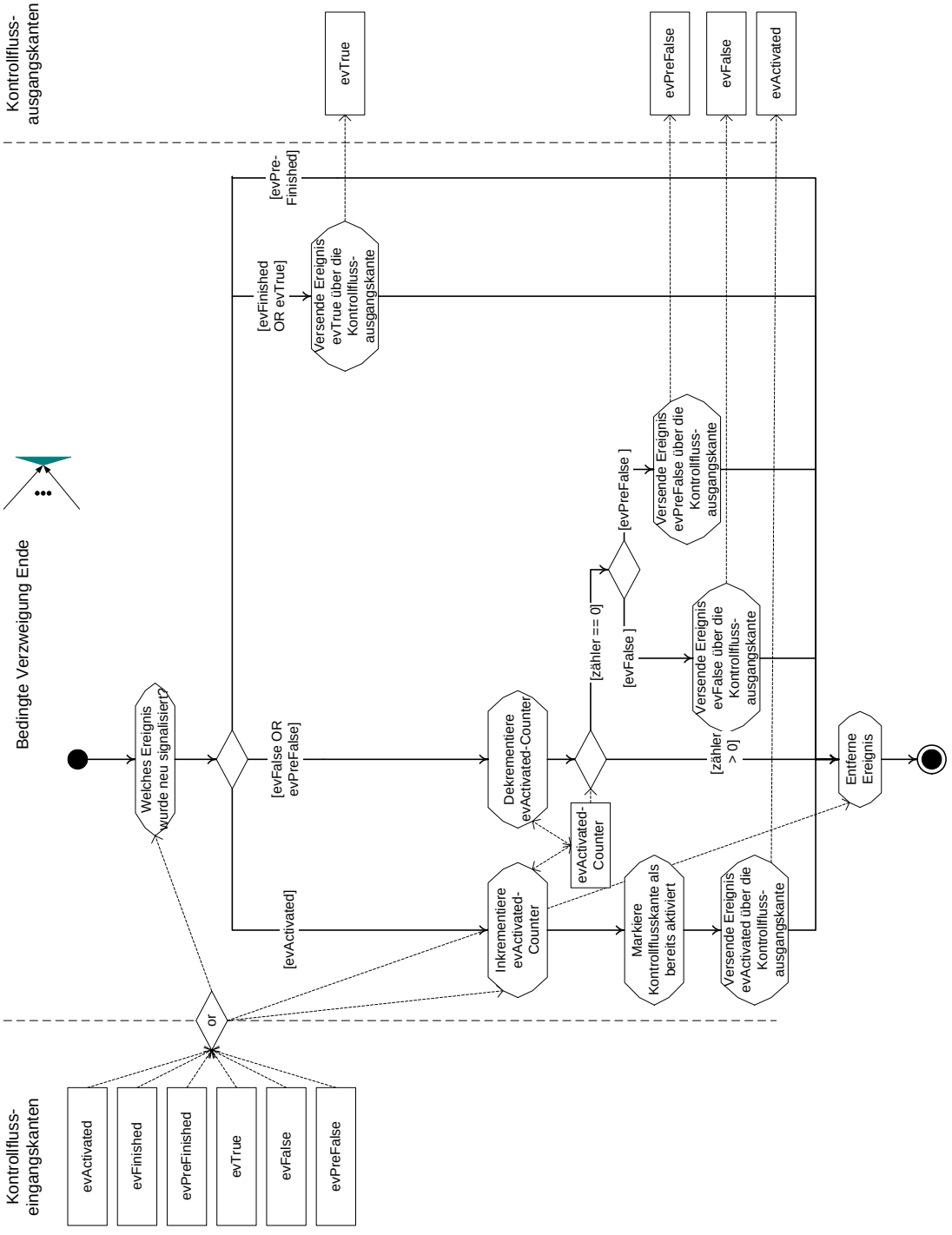



**Abbildung B.5:** Schaltlogik des Kontrollflussknotens Verzweigung Ende ◄

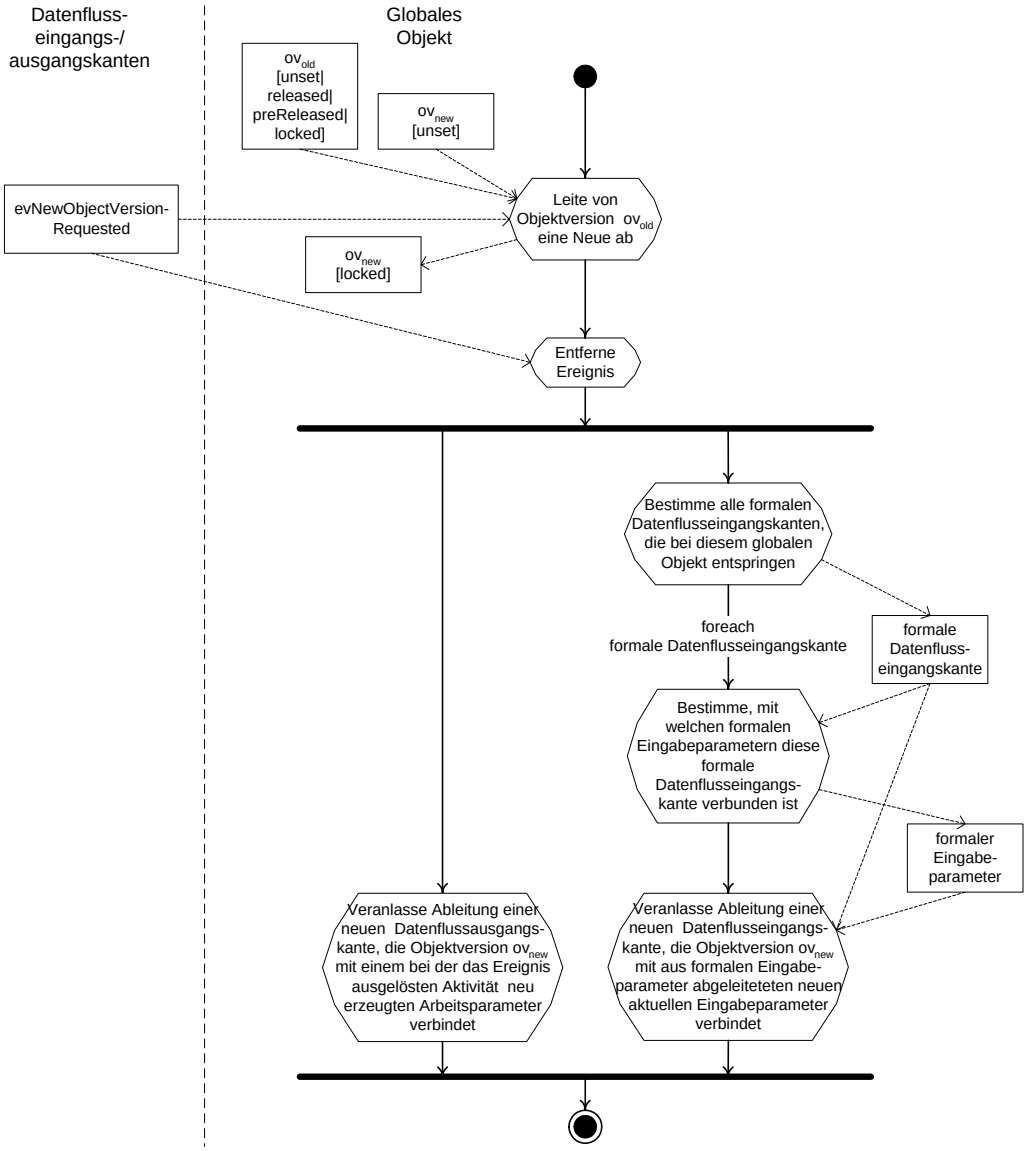



**Abbildung B.6:** Verhalten eines globalen Objekts beim Ereignis $evNewObjectVersionRequested$

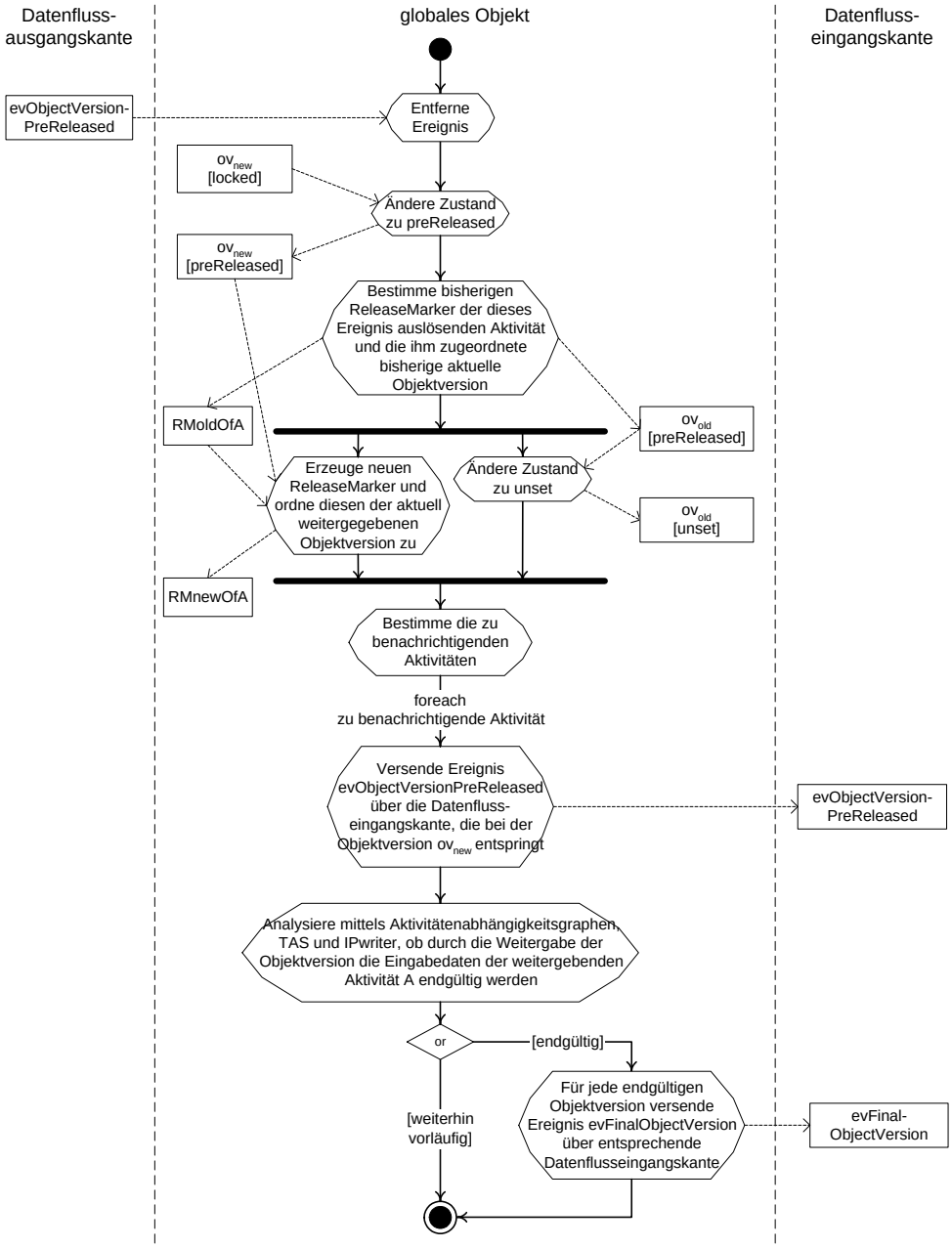



**Abbildung B.7:** Verhalten einer Objektversion beim Ereignis $evObjectVersionPreReleased$

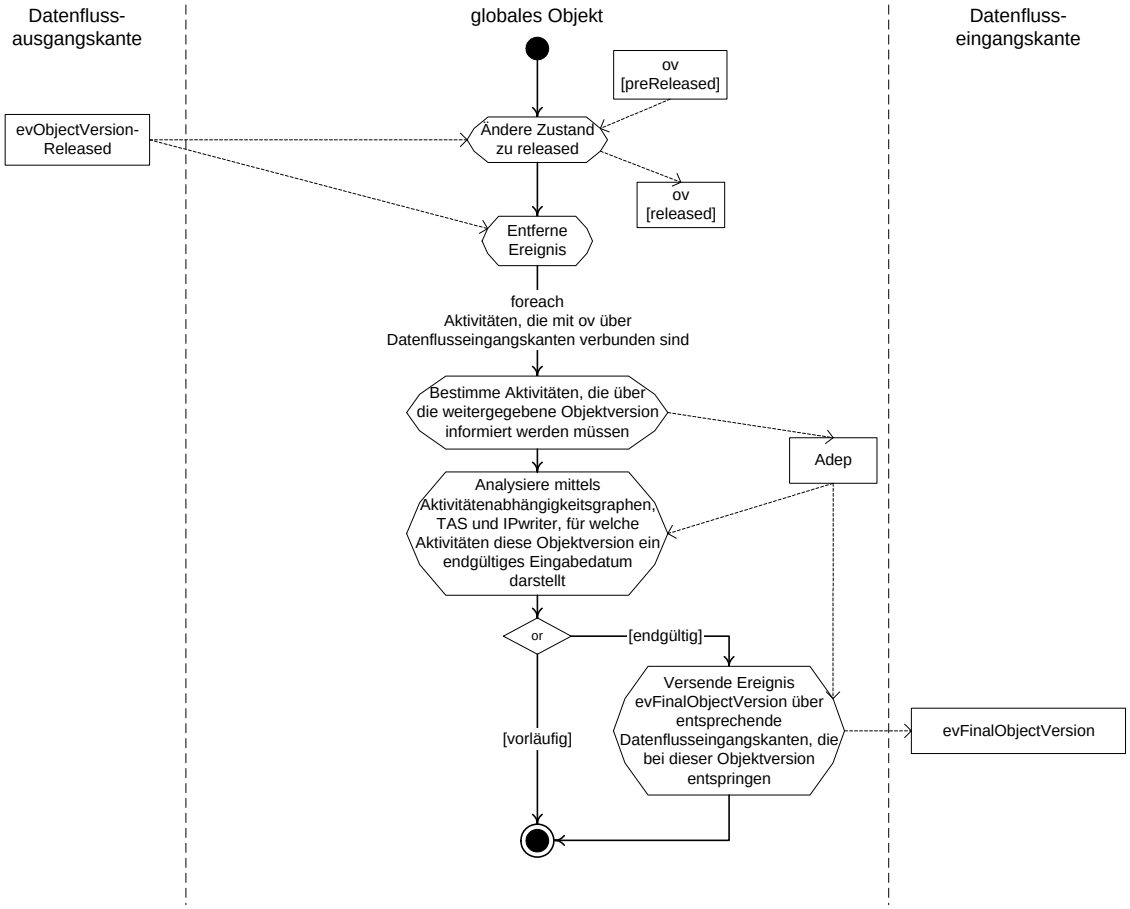



**Abbildung B.8:** Verhalten einer Objektversion beim Ereignis $evObjectVersionReleased$: $TAS$ (= *Terminated Activities Set*) und $IPwriter$ sind Hilfsstrukturen, die zur Bestimmung der Endgültigkeit einer Objektversion benötigt werden (vergleiche Abschnitt 13.6.3).

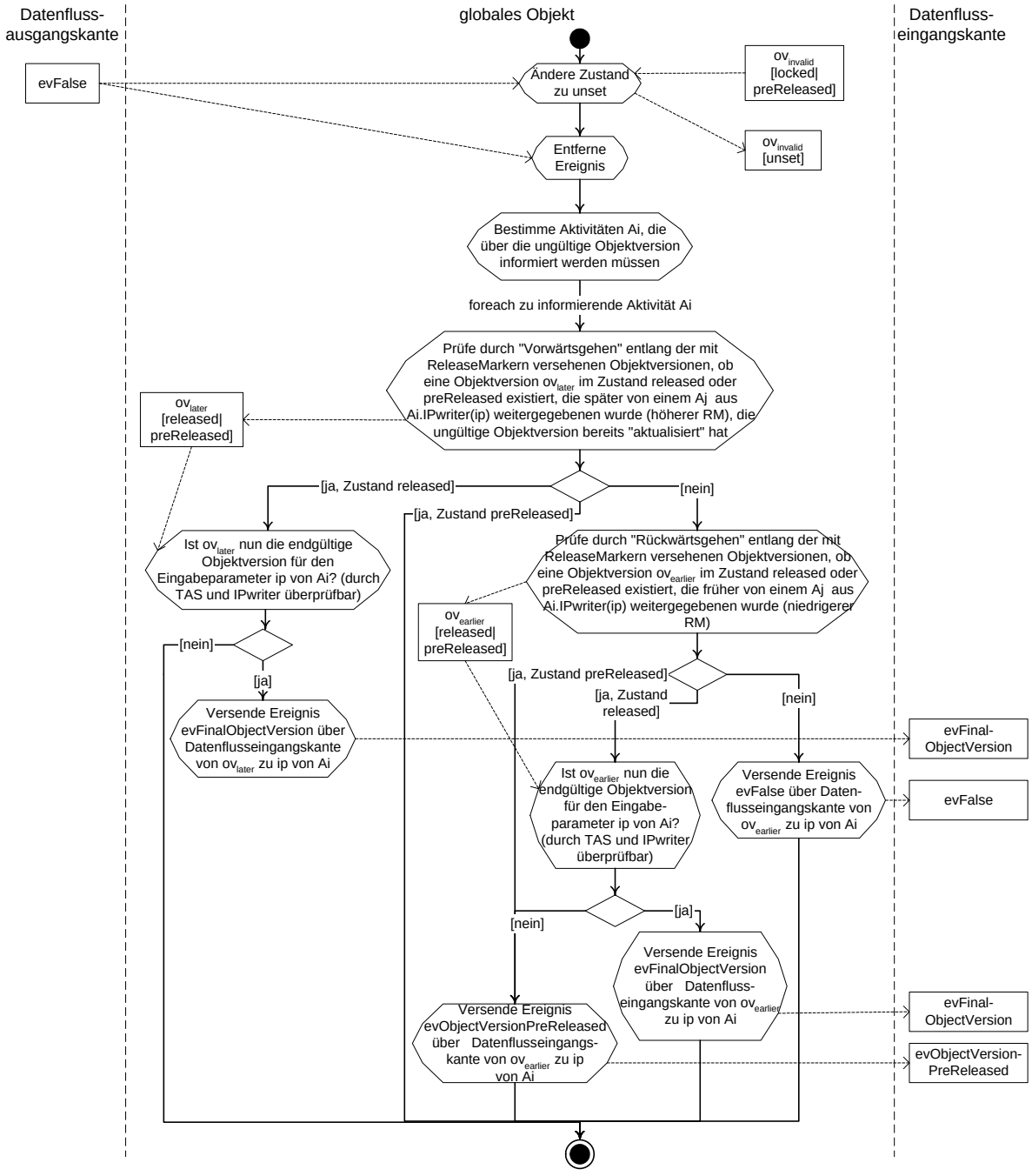



**Abbildung B.9:** Verhalten einer Objektversion beim Ereignis $evFalse$: $TAS$ (= *Terminated Activities Set*) und $IPwriter$ sind Hilfsstrukturen, die zur Bestimmung der Endgültigkeit einer Objektversion benötigt werden (vergleiche Abschnitt 13.6.3).

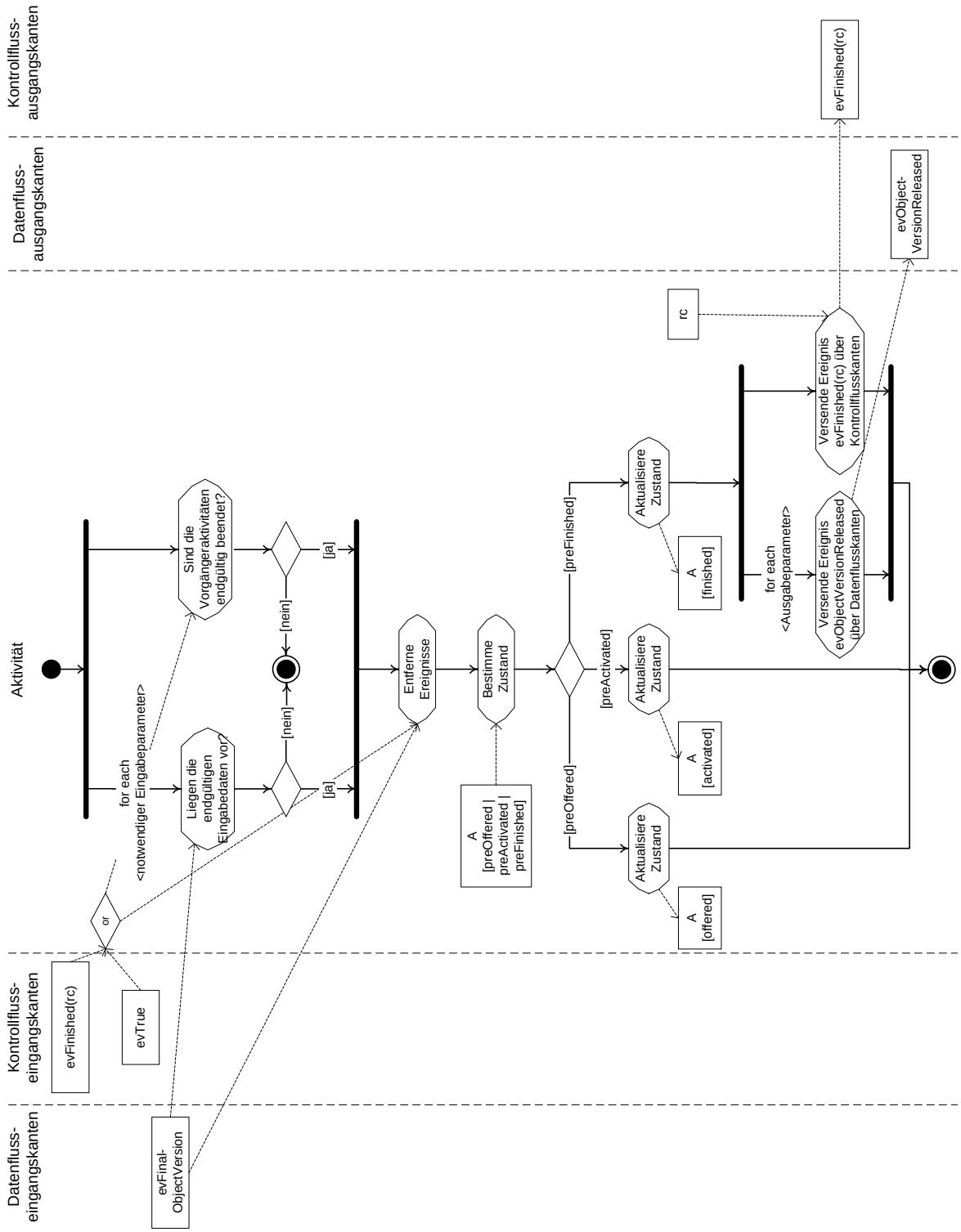



**Abbildung B.10:** Verhalten einer abhängigen Aktivität beim Übergang zur Unabhängigkeit

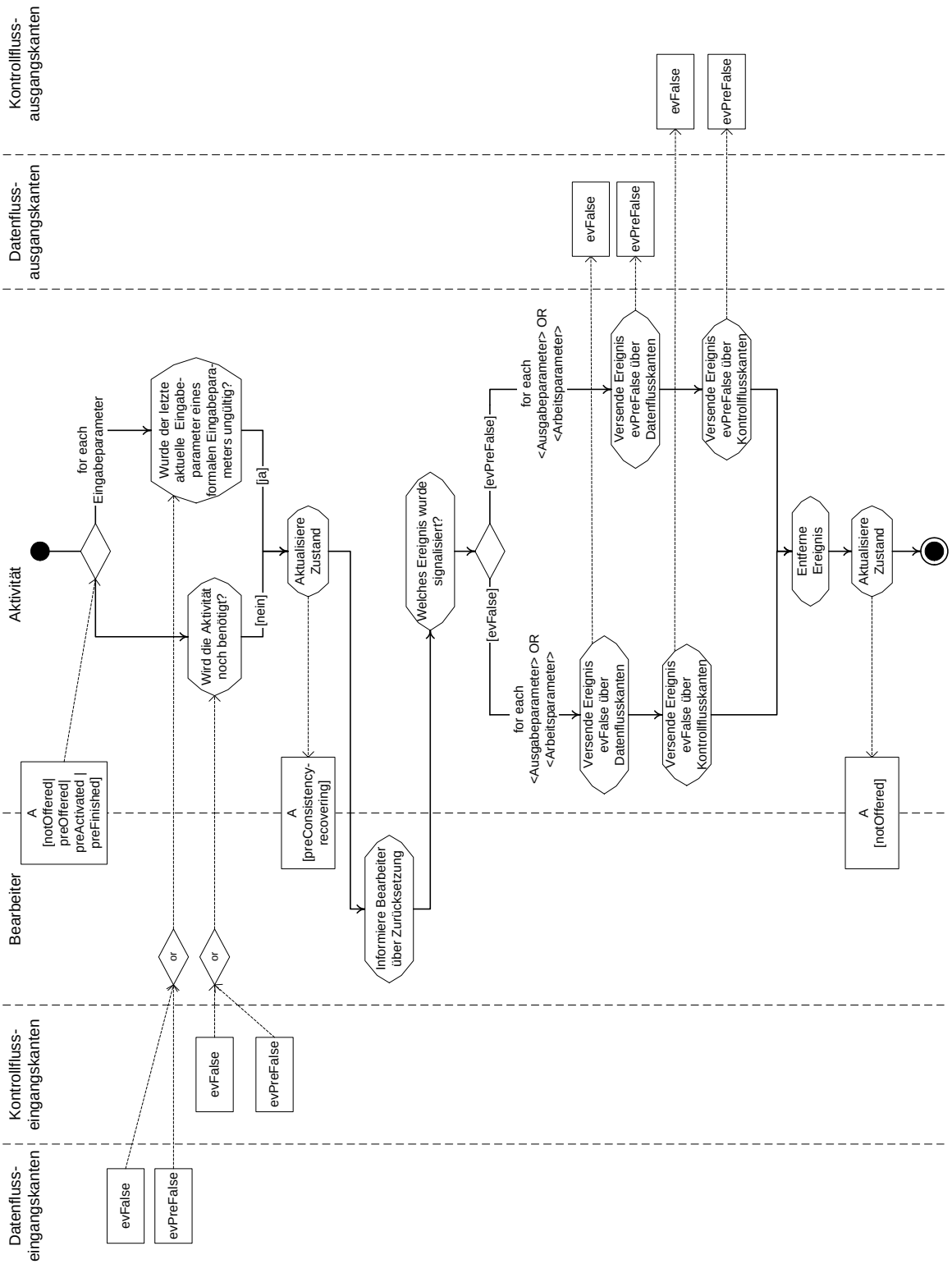



**Abbildung B.11:** Verhalten einer abhängigen Aktivität bei der Undo-Konsistenzsicherung

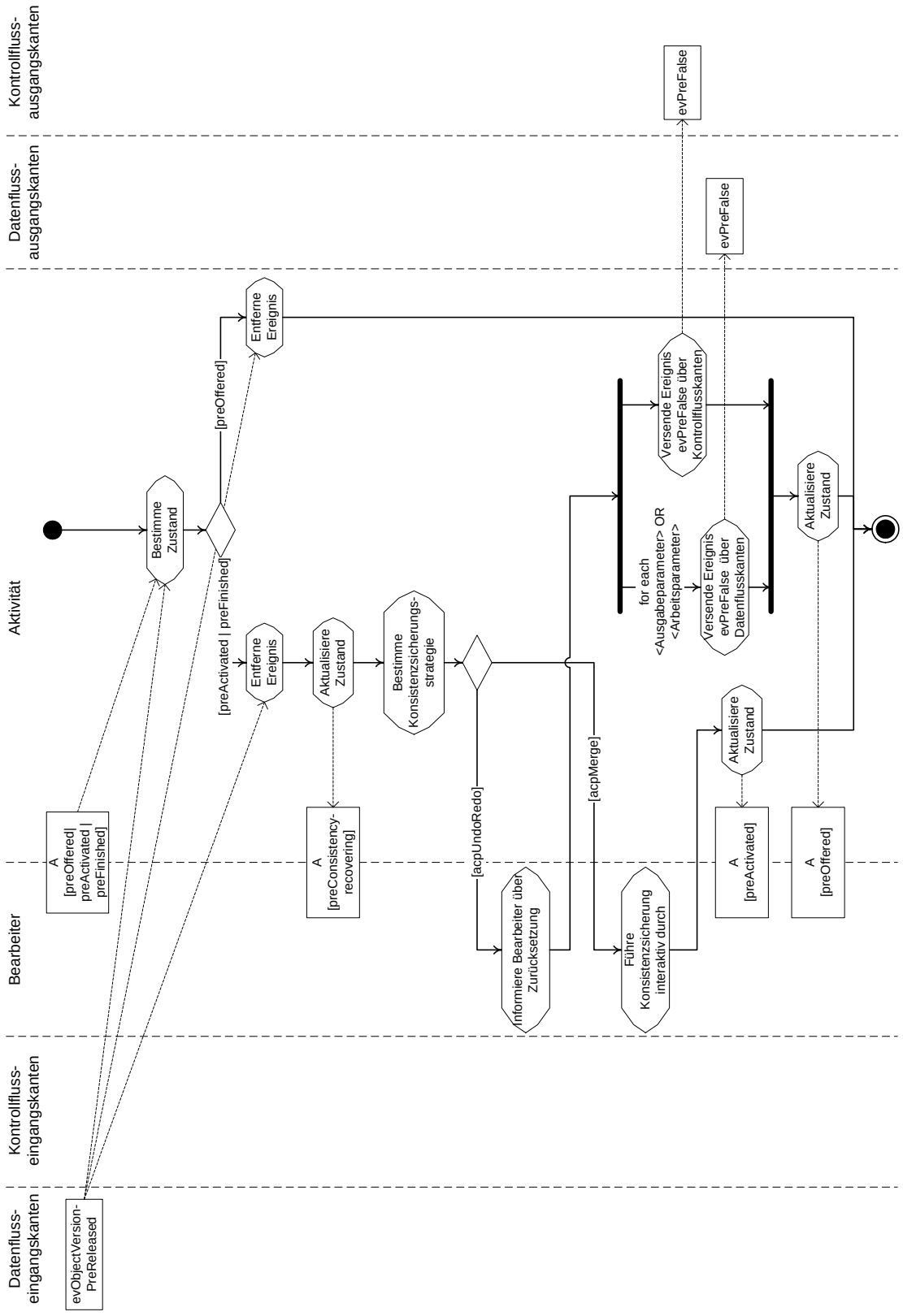



**Abbildung B.12:** Verhalten einer abhängigen Aktivität beim Eintreffen neuer Eingabedaten

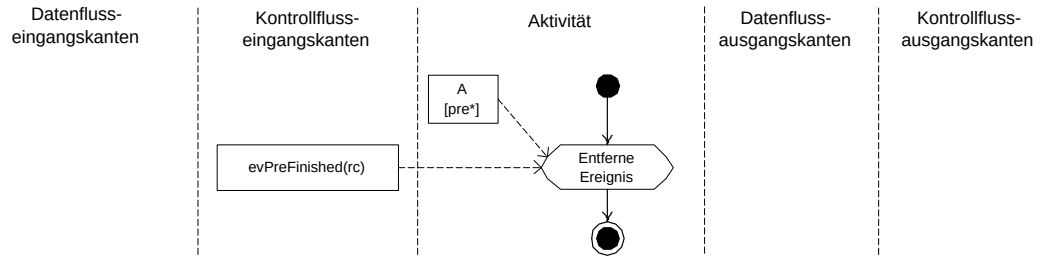



**Abbildung B.13:** Verhalten einer abhängigen Aktivität beim Ereignis $evPreFinished(rc)$