Universität Leipzig

Fakultät für Mathematik und Informatik

Institut für Informatik

Konzeption und Implementierung eines Agenten zur ereignisorientierten Adaptation von Workflows

Diplomarbeit

Aufgabenstellung und Betreuung:
Prof. Dr. habil. E. Rahm,
Dipl.-Math. R. Müller

vorgelegt von:
Ulrike Greiner

Leipzig, Juli 2000

Vorwort

Im Rahmen des Projektes HEMATOWORK wurde das Workflow-Management-System AGENTWORK entwickelt, das die Möglichkeit bietet, Workflows zur Laufzeit dynamisch zu adaptieren. AGENTWORK wurde im Rahmen des medizinischen Kontextes von HEMATOWORK entwickelt, ist aber auch in anderen Anwendungsgebieten einsetzbar.
Das Thema der vorliegenden Arbeit, der sogenannte Adaptations-Agent, bildet eine Komponente des Systems AGENTWORK.

Folgenden Personen möchte ich danken:

- Herrn Prof. Dr. E. Rahm für die Aufgabenstellung und die Unterstützung der Arbeit,
- Herrn Prof. Dr. A. Winter für die hilfreichen Hinweise,
- Herrn R. Müller für die umfassende Betreuung der Arbeit,
- meinen Kommilitonen Alexander Dietzsch und Rainer Böhme für die anregenden Diskussionen.

Inhaltsverzeichnis

1. Einleitung

Die vorliegende Arbeit ist Teil der Projekte HEMATOWORK und AGENTWORK, die vom Institut für Informatik und dem Institut für Medizinische Informatik, Statistik und Epidemiologie der Universität Leipzig gemeinsam durchgeführt werden. Ziel ist es, die Durchführung von Therapiestudien (beispielsweise zu Chemotherapien oder Radiotherapien) in der Hämato-Onkologie durch den Einsatz von Workflow-Management-Systemen zu unterstützen. Insbesondere soll der behandelnde Arzt bei der Durchführung der Therapie unterstützt werden. Zusätzlich soll die Kommunikation zwischen den beteiligten Institutionen verbessert, die Vollständigkeit der Dokumentation gesichert und damit die Qualität der Studienergebnisse erhöht werden.

Während der Durchführung von Chemotherapien oder Radiotherapien können unvorhergesehene Ereignisse eintreten, die keine technischen Fehler (wie z. B. Rechnerausfälle) sind, sondern die Ablauflogik der Behandlung betreffen und eine Änderung der Therapie nötig machen. Dazu zählen z. B. eine plötzlich auftretende Allergie gegen einen bestimmten Wirkstoff oder eine Infektion. Die temporale Auswirkung solcher Ereignisse kann sehr unterschiedlich sein. Während bei einer Allergie der Patient einen bestimmten Wirkstoff nie mehr bekommen darf und deshalb die laufende und alle weiteren Behandlungen von diesem Ereignis betroffen sind, kann es bei einer Infektion ausreichend sein, die Chemotherapie für eine bestimmte Zeit auszusetzen oder durch zusätzliche Medikamente zu ergänzen. Das Infektions-Ereignis wirkt sich also nur auf einen zeitlich begrenzten Teil der Behandlung aus.

Ein Workflow-System, das die Behandlung sinnvoll unterstützen soll, muss deshalb auf die verschiedenen Arten von Ereignissen reagieren können. Einerseits muss es bei technischen Fehlern die weitere Durchführung der Behandlung sicherstellen, indem es z. B. den Administrator einer ausgefallenen Datenbank oder eines abgestürzten Rechners informiert. Andererseits muss es in der Lage sein, beim Eintritt eines Ereignisses, das die Ablauflogik der Behandlung betrifft, den Workflow vollständig abzubrechen oder ihn zu unterbrechen, abzuändern oder zu ergänzen.

Da verfügbare kommerzielle Systeme diese Möglichkeit nur eingeschränkt bieten, wird im Rahmen der beiden Projekte das Workflow-Management-System AGENTWORK entwickelt, das auf Ereignisse reagieren und den betroffenen Workflow zur Laufzeit an die neue Situation anpassen kann.

Im Folgenden wird der medizinische Hintergrund der Projekte HEMATOWORK und AGENT-WORK dargestellt. Anschließend wird erklärt, was Workflow-Management-Systeme sind, wie sie arbeiten und wie sie zur Unterstützung von Therapiestudien in der Hämato-Onkologie eingesetzt werden können. Danach werden die Ziele der vorliegenden Arbeit abgesteckt und kurz bisherige Ansätze untersucht. Zum Abschluss dieses Kapitels wird der weitere Aufbau der Arbeit erläutert.

1.1 Medizinischer Hintergrund - Hämato-Onkologie

Die Hämato-Onkologie befasst sich mit der Erforschung und Behandlung von sogenannten nicht-soliden Tumoren wie Lymphomen oder Leukämien ([Die98], [NHL94]). Da diese eine häufige Todesursache sind, beschäftigen sich viele Forscher mit der Entwicklung neuer, nebenwirkungsärmerer Therapien. Um ihre Wirksamkeit zu zeigen, werden sie ausgiebig getestet. Dies erfolgt im Rahmen von klinischen Studien, bei denen verschiedene Therapien miteinander verglichen werden ([Mic97]). Dabei werden alle Ergebnisse dokumentiert und anschließend statistisch ausgewertet.
Für den Gesamtablauf einer Studie sowie für die einzelnen Therapien gibt es standardisierte Behandlungspläne. In Abbildung 1-1 ist der Behandlungsplan für die Studie B bei hochmalignem NHL dargestellt, dessen jeweils umfangreich dokumentierte Schritte im Folgenden kurz erläutert werden. Dabei werden zum besseren Verständnis die Bezeichnungen aus der Abbildung jeweils in Klammern angeführt.

Nach der Aufnahme eines Patienten in die Studie (*Meldung*) werden zuerst initiale Untersuchungen (*Staging*) und anschließend eine Toxizitäts-Untersuchung durchgeführt (*Tox-Untersuchung*). Danach wird der Patient zufällig einer der vier unterschiedlichen Chemotherapien zugewiesen (*Randomisation*). Anschließend wird die ausgewählte Chemotherapie durchgeführt (*CHOP 21 / CHOEP 21 / CHOP 14 / CHOEP 14*). CHOP 21 bezeichnet dabei die Standardtherapie mit Therapiezyklen von 21 Tagen. Bei CHOP 14 werden die Therapiezyklen von 21 auf 14 Tage verkürzt. Bei den CHOEP-Therapien wird zusätzlich Etoposid gegeben.
Danach werden erneut Untersuchungen durchgeführt (*Zwischen-Restaging*), deren Ergebnisse den weiteren Fortgang der Behandlung bestimmen. Im Falle einer kompletten oder partiellen

Tumorremission wird die Behandlung mit der gewählten Chemotherapie fortgesetzt (*CHOP 21 / CHOEP 21 / CHOP 14 / CHOEP 14*). Sollte die Chemotherapie kaum oder gar nicht ansprechen, wird eine sogenannte Salvage-Therapie durchgeführt (*Salvage-Protocol*).
Nach dem Ende der zweiten Chemotherapie-Behandlung wird eine abschließende Untersuchung durchgeführt (*Restaging*), deren Befund wiederum unterschiedliche Folgeschritte veranlassen kann. Ist bei einem Patienten eine komplette Tumorremission eingetreten und der Patient hat keine befallenen Lymphknoten mehr, muss er in bestimmten Abständen zu Nachsorge und Toxizitäts-Untersuchungen kommen (*Nachsorge, Tox-Untersuchung)*. Ein Patient mit kompletter Tumorremission, jedoch einem zu Beginn der Therapie stark befallenen Lymphknoten wird noch einer zusätzlichen Strahlentherapie (*Strahlentherapie*) mit anschließender Untersuchung (*Zusatz Restaging*) unterzogen. Danach muss er ebenfalls regelmäßig zu Nachsorge und Toxizitäts-Untersuchungen kommen (*Nachsorge, Tox-Untersuchung*). Hat sich der Tumor nicht verkleinert, wird eine Salvage-Therapie durchgeführt (*Salvage-Protocol*).

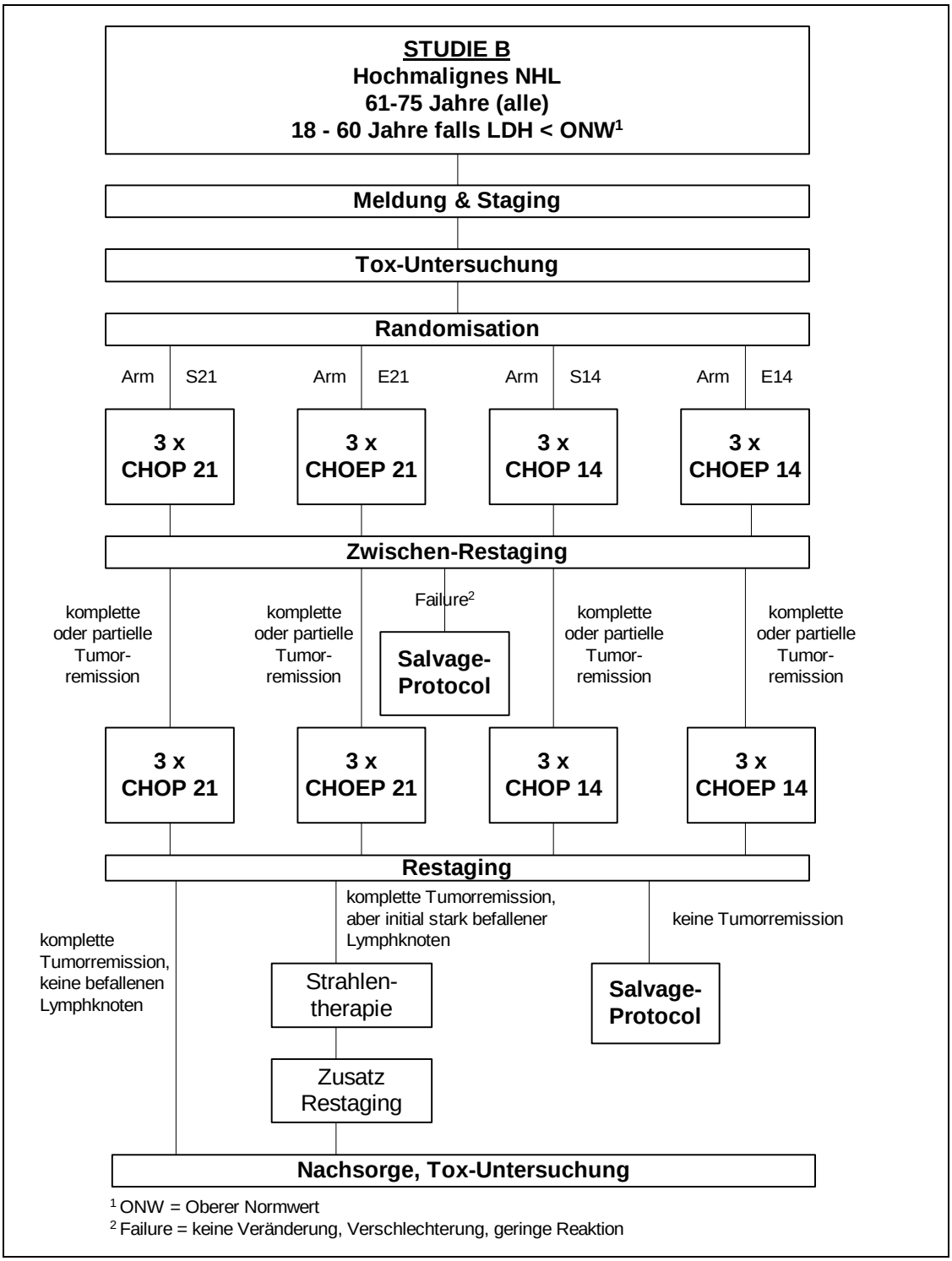


Abb. 1-1: Hochmalignes NHL - Behandlungsplan für die Studie B (nach [NHL94]).

1.2 Technischer Hintergrund - Workflow-Management-Systeme

Im Folgenden wird definiert, was ein Workflow-Management-System ist und wie es arbeitet. Dabei werden wichtige Begriffe erläutert, die in der vorliegenden Arbeit verwendet werden. Der Zusammenhang zwischen diesen Begriffen ist in Abbildung 1-2 illustriert.

Arbeitsabläufe in einem Unternehmen (z. B. Reisebuchung, Bearbeitung einer Kundenbestellung) oder in der Medizin (z. B. Chemotherapie) können als Anwendungsprozesse beschrieben werden, d. h. als eine Menge von zusammengehörigen Aktivitäten, die das Erreichen eines bestimmten Zieles realisieren. Die Aktivitäten können dabei manuell oder automatisiert sein. Im ersten Fall werden sie von einem Benutzer durchgeführt, im zweiten Fall gibt es Applikationen, die diese (eventuell in Interaktion mit einem Benutzer) ausführen.
Wenn diese Prozesse stark strukturiert sind, also die Reihenfolge der Aktivitäten festen Regeln folgt, kann man sie in einer Workflow-Definition (kurz einem Workflow) modellieren. Diese besteht aus dem Kontrollfluss (die Aktivitäten des Anwendungsprozesses und die Beziehungen zwischen ihnen), dem Datenfluss (der Weg von Dokumenten und Informationen zwischen Aktivitäten) und zusätzlichen Informationen zu den einzelnen Aktivitäten. Zusätzliche Informationen können die Bezeichnung der zu startenden Applikation oder der Name des zuständigen Sachbearbeiters sein. Werden der Kontroll- und Datenfluss grafisch dargestellt, verbessert das die Übersichtlichkeit und Fehler (z. B. in der logischen Abfolge der einzelnen Schritte eines Anwendungsprozesses) können leichter erkannt werden. Wird für die grafische Darstellung ein Graphmodell mit Knoten und Kanten gewählt, können beispielsweise die Aktivitäten der Workflow-Definition den Knoten und die Beziehungen zwischen ihnen den Kanten zugeordnet werden.
Soll ein bestimmter Prozess, zum Beispiel eine Reisebuchung, durchgeführt werden, wird eine Instanz von der entsprechenden Workflow-Definition erzeugt und ausgeführt. Dabei werden die einzelnen Aktivitäten des Kontrollflusses nacheinander abgearbeitet. Das umfasst die Bereitstellung der benötigten Daten und das Starten einer Applikation bei einer automatischen Aktivität bzw. die Benachrichtigung eines Sachbearbeiters bei einer manuellen oder halbautomatischen Aktivität.

Ein System, das die Modellierung von Geschäftsprozessen in Workflows mit Kontroll- und Datenflüssen ermöglicht, die verschiedenen Workflows verwaltet, Instanzen erzeugt und diese ausführt, bezeichnet man als Workflow-Management-System. [vgl. WfMC99, S. 7-17]

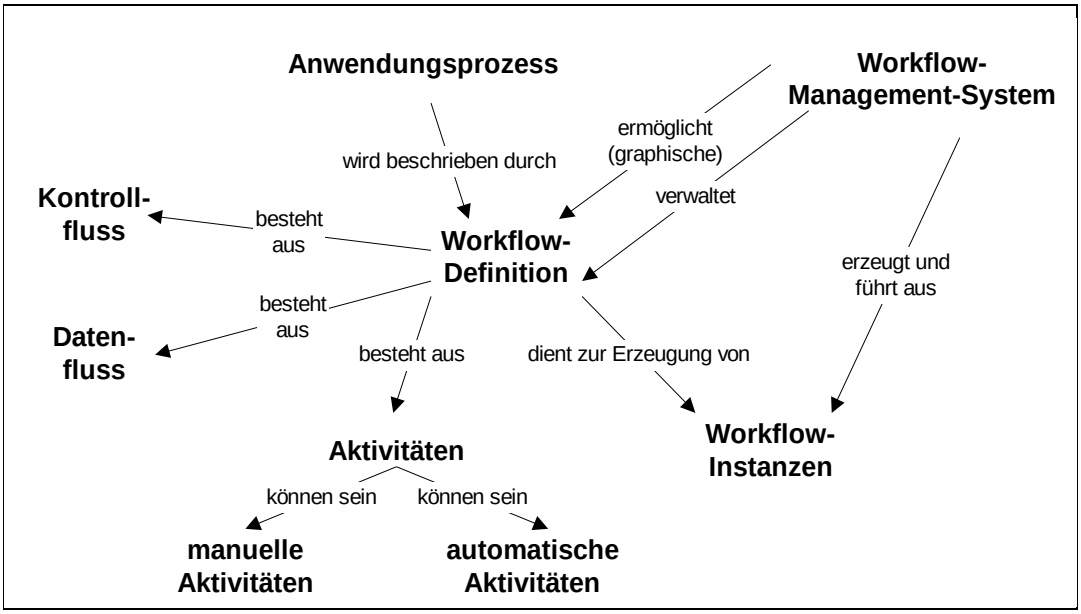


Abb. 1-2: Wichtige Workflow-Management-Begriffe und ihr Zusammenhang (nach [WfMC99], S.7).

Ein Workflow-Management-System besteht aus zwei Hauptteilen: einem *Workflow-Editor* zur Definition von Workflows und einer *Workflow-Engine*, die die Abarbeitung der einzelnen Workflows steuert und überwacht. Dazu erstellt die Workflow-Engine eine Instanz der Workflow-Definition und arbeitet diese ab. Ist dabei eine Aktivität auszuführen, startet die Engine entweder direkt eine Applikation (automatische Aktivität) oder benachrichtigt den für diese Aktivität zuständigen Workflow-Client (manuelle Aktivität). Das ist ein Programm, das z. B. auf dem Rechner des zuständigen Sachbearbeiters läuft und diesen informiert, wenn eine Aktivität durchzuführen ist.

Die Definition von Workflows mit Hilfe des Editors bezeichnet man auch als *Buildtime*, die Ausführung durch die Engine als *Runtime.*

In Abbildung 1-3 sind die Grundkomponenten eines Workflow-Management-Systems und ihre Schnittstellen dargestellt.

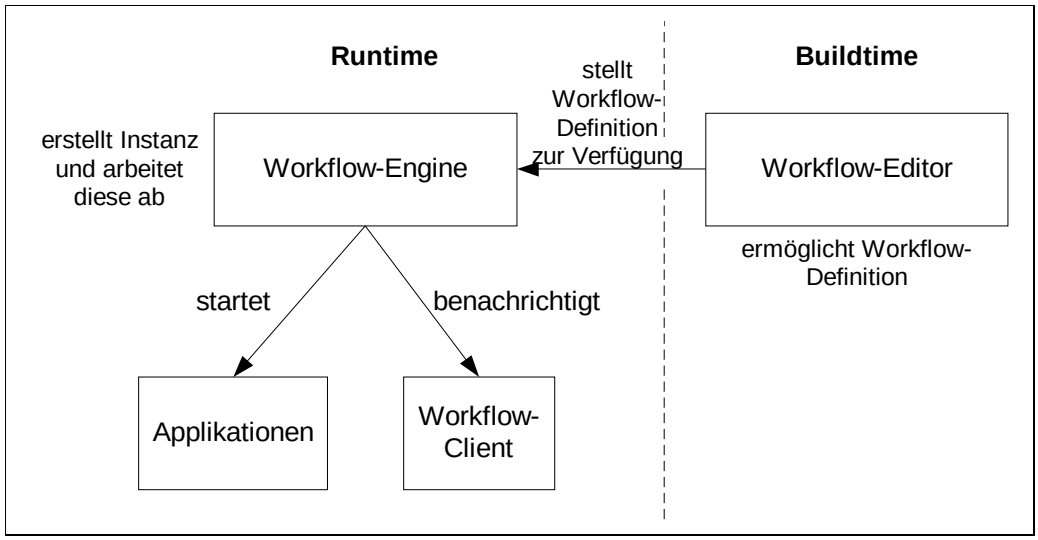


Abb. 1-3: Kernarchitektur eines Workflow-Management-Systems (nach [WfMC99]).

Die Behandlungspläne für Therapiestudien in der Hämato-Onkologie bilden einen komplexen Anwendungsprozess strukturiert ab (vgl. Abb. 1-1). Damit können sie als Grundlage für die Definition von Workflows dienen. Man kann also jeden Behandlungsplan als Vorstufe eines Workflows auffassen.

Wie die Umsetzung eines solchen Plans in einen Workflow konkret aussehen kann, wird in Abbildung 1-4 illustriert. Der dargestellte Workflow bildet den Ablauf eines Zyklus der CHOP 14-Chemotherapie ab. In diesem sind am ersten Tag alle vier Medikamente (Cyclophosphamid, Doxorubicin, Vincristin und Prednison) zu geben. An den Tagen zwei bis fünf ist dann nur noch Prednison zu verabreichen. Anschließend wird noch ein Report an die Studienkommission gesendet.

Durch den AND-Split, bei dem alle Pfade gleichzeitig durchlaufen werden, wird ausgedrückt, dass am ersten Tag alle Medikamente gegeben werden. Die wiederholte Gabe von Prednison wird durch die Schleife modelliert, indem beim Loop-Start-Knoten ein Zähler mit 1 initialisiert wird (i:=1). Nach Durchlaufen der Schleife wird dieser Zähler um eins inkrementiert (i++) und danach verglichen, ob der neue Wert kleiner 6 ist. Wenn ja, wird die Schleife ein weiteres Mal durchlaufen, wenn nein, sind die fünf Tage vorbei und die Schleife wird endgültig verlassen. Damit hat auch der letzte Pfad den AND-Join erreicht (die ersten drei Pfade dauerten jeweils nur einen Tag) und die nächste Aktivität (Sende Report) kann ausgeführt werden. Wenn diese zu Ende ist, ist das Ende des Workflows erreicht.

Die Angabe *Verabreiche Medikament X* ist verkürzt, um die Abbildung übersichtlicher zu machen. Für eine vollständige Spezifikation müssten beispielsweise noch die Dosis und die Art der Verabreichung mitangegeben werden.

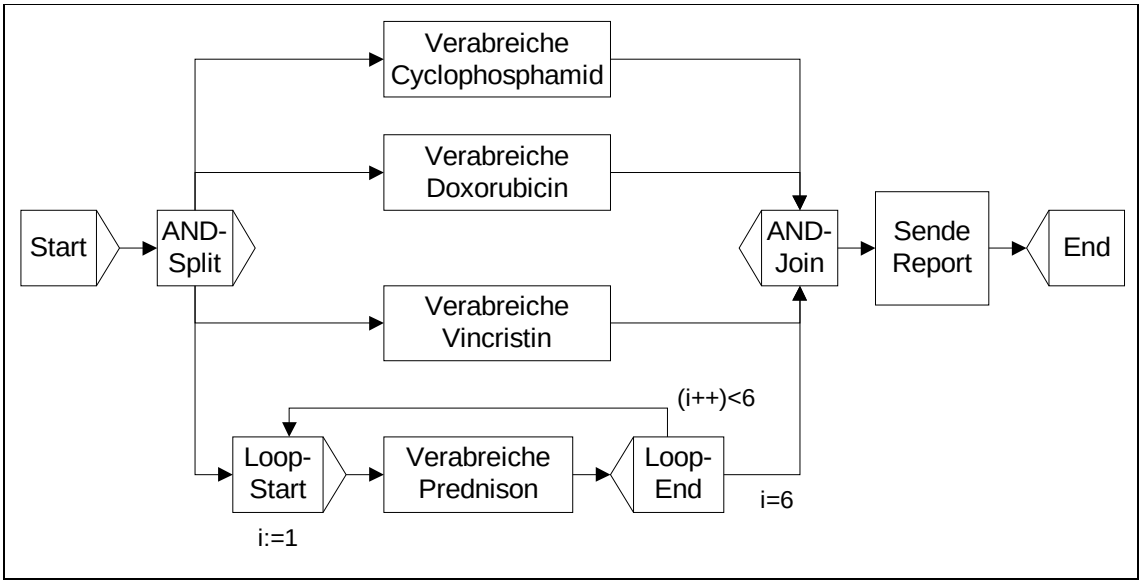


Abb. 1-4: CHOP 14 - Workflow (nach [NHL94]).

Während der Durchführung der Chemotherapie kann es zu unerwarteten Ereignissen wie z. B. einer Infektion oder Nebenwirkungen kommen. Je nach der temporalen Auswirkung des Ereignisses muss der Workflow verändert werden. Wenn das Ereignis z. B. nur die Tage zwei und drei betrifft, ist nur der Teil betroffen, der an diesen beiden Tagen ausgeführt wird. Im Beispiel wären das die beiden Schleifendurchläufe mit den Zählerwerten i=2 und i=3. Danach kann die Therapie in ihrer ursprünglichen Form fortgesetzt werden.
Ist die zeitliche Auswirkung größer und reicht z. B. bei einem Ereignis, das am Tag zwei eintritt, über fünf Tage, ist der ganze restliche Workflow bis zum End-Knoten davon betroffen und muss entsprechend geändert werden.

1.3 Problemstellung und Zielsetzung der Arbeit

Ziel der vorliegenden Arbeit ist es, eine Teilkomponente des Workflow-Systems AGENTWORK zu entwickeln, den sogenannten Adaptations-Agenten. Dieser führt die Änderungen an einem von einem Anwendungsereignis (keinem technischen Fehler) betroffenen Workflow durch.

Ein Anwendungsereignis kann z. B. eine Neuropathie (Nervenschädigung) des behandelten Patienten sein, weshalb das Medikament Vincristin nicht mehr verabreicht werden darf. Der in Abbildung 1-4 dargestellte CHOP 14-Workflow muss also so verändert werden, dass die Aktivität *Verabreiche Vincristin* nicht mehr darin vorkommt. Das Ergebnis dieser Adaptation zeigt Abbildung 1-5. Der Knoten, der die Aktivität *Verabreiche Vincristin* repräsentiert, wurde gelöscht. Da der Splitpfad, der diese Aktivität enthielt, keine weiteren Aktivitäten enthält, hat der AND-Split jetzt nur noch drei Pfade.

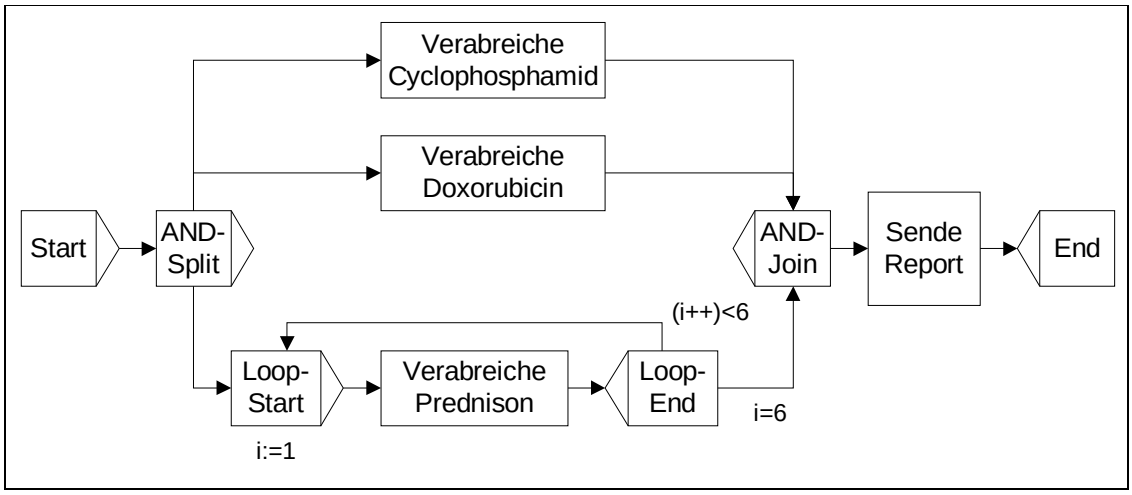


Abb. 1-5: Adaptierter CHOP 14 - Workflow.

Bei der Ausführung eines Workflows können zwei Arten von Fehlern auftreten, nämlich *technische* und *logische* Fehler. Technische Fehler entstehen z. B. wenn eine Datenbank, ein Rechner oder Teile eines Netzwerkes ausfallen und die Ausführung des Workflows dadurch behindert wird.

Logische Fehler werden nach [Mue00] wie folgt definiert:

> Ein *logischer Workflow-Fehler* liegt dann vor, wenn aufgrund eines Anwendungsereignisses der Kontrollfluss eines Workflows nicht mehr adäquat ist.

Ist während der Ausführung einer Workflow-Instanz ein logischer Fehler aufgetreten, muss der Kontrollfluss verändert werden. Dazu müssen entweder wie im obigen Beispiel Knoten gelöscht werden, oder sie werden ersetzt, verzögert oder neu eingefügt (beispielweise zur Behandlung von Nebenwirkungen). Damit dem Adaptations-Agenten, der die Änderungen am Kontrollfluss vornimmt, bekannt ist, welche Knoten zu entfernen, zu ersetzen, zu verzögern oder zusätzlich einzufügen sind, werden mit Hilfe von Regelbasen sogenannte *lokale Kontroll-Aktionen* spezifiziert:

> *Kontroll-Aktionen* sind Arbeitsanweisungen für einen Agenten.
>
> *Lokale Kontroll-Aktionen* enthalten den von einer Adaptation betroffenen Aktivitätstyp (z. B. verabreiche das Medikament X), die durchzuführende Aktion (entfernen, verzögern, ersetzen, zusätzlich ausführen) und das Gültigkeitsintervall *T* der Kontroll-Aktion.
>
> *Globale Kontroll-Aktionen* betreffen einen ganzen Workflow. Sie enthalten die durchzuführende Aktion (suspendieren, abbrechen) und eine eindeutige Bezeichnung des betroffenen Workflows.

Die Konsistenz der Regelbasen wird mit bekannten Verfahren gesichert. Dies gehört nicht zu den Aufgaben des Adaptations-Agenten und wird deshalb in der vorliegenden Arbeit nicht weiter behandelt (siehe [PP00, SSS82] als weiterführende Referenzen).
Für den Adaptations-Agenten sind nur lokale Kontroll-Aktionen von Bedeutung. Deshalb sind im Folgenden immer lokale Kontroll-Aktionen gemeint, wenn von Kontroll-Aktionen die Rede ist. Sind globale Kontroll-Aktionen gemeint, wird das explizit angegeben.

Das Gültigkeitsintervall *T* ist ein Zeitintervall, das angibt, wie lange die Kontroll-Aktion gilt. Es ergibt sich aus der temporalen Auswirkung des zugehörigen Anwendungsereignisses. Die temporale Auswirkung kann z. B. bei einer Allergie unendlich sein, sie kann aber auch nur einige Stunden oder Tage umfassen, wenn der Patient z. B. eine Infektion bekommen hat und deswegen für einige Tage ein zusätzliches Antibiotikum benötigt. Die temporalen Auswirkungen von Anwendungsereignissen und damit von Kontroll-Aktionen können also sehr unterschiedlich sein. Damit sind auch die Teile des Workflows, die von einem Ereignis betroffen sind, unterschiedlich und für jedes Ereignis anders.

Die erste Hauptaufgabe des Agenten ist es deshalb, den Teil eines Workflow zu bestimmen, der während der Zeit, auf die das Ereignis Auswirkungen hat, voraussichtlich durchlaufen werden wird. Dieser Teil-Workflow wird als *Adaptations-Region* bezeichnet und ist der Teil des Workflows, in dem die durch das Ereignis bedingten Änderungen durchzuführen sind.
Tritt z. B. bei einem Patienten eine Allergie gegen einen bestimmten Wirkstoff auf, wirkt sich diese auf den ganzen restlichen Workflow und auf alle zukünftig für diesen Patienten gestarteten Workflows aus, da der Patient diesen Wirkstoff nie mehr bekommen darf. Die Adaptations-Region für dieses Ereignis beginnt also an der Stelle des Workflows, die gerade ausgeführt wurde, als die Allergie festgestellt wurde, und reicht bis zum Ende des Workflows. Werden für diesen Patienten weitere Workflows gestartet, gehören auch sie vom Start- bis zum Ende-Knoten zur Adaptations-Region.
Ist die Auswirkung eines Ereignisses zeitlich begrenzt, wie z. B. bei einer Infektion, gehört nur der Teil des Workflows zur Adaptations-Region, der bis zum Abklingen der Infektion durchlaufen wird. In einem während der Infektion neu gestarteten Workflow gehört der Teil vom Start-Knoten bis zu dem Knoten zur Adaptations-Region, an dem die Infektion ausgeheilt ist.
In Abbildung 1-6 ist dieser Fall eines in seiner Auswirkung zeitlich begrenzten Ereignisses noch einmal illustriert. Das Ereignis tritt ein, während der Workflow 1 läuft. Zusätzlich wird während des Gültigkeitsintervalls des Ereignisses ein zweiter Workflow (Workflow 2) gestartet, der auch von diesem Ereignis betroffen ist. Die Teile der beiden Workflows, die während der zeitlichen Auswirkung des Ereignisses durchlaufen werden, bilden die Adaptations-Region für dieses Ereignis.

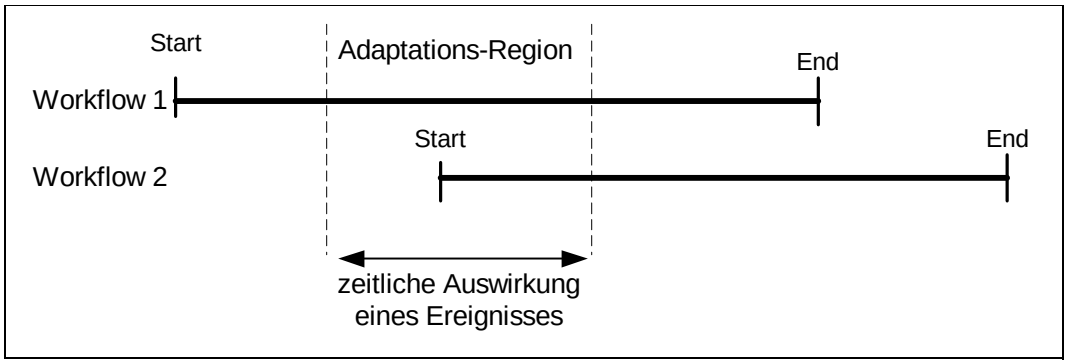


Abb. 1-6: Begrenzte zeitliche Auswirkung eines Ereignisses.

Hat der Agent eine Adaptations-Region bestimmt, muss er darin alle Änderungen durchführen, die durch das Ereignis erforderlich geworden sind. Diese Änderungen beziehen sich immer auf eine Instanz einer Workflow-Definition. Die Workflow-Definition selbst wird nicht geändert. Sie bleibt im ursprünglichen Zustand erhalten, da sie als Vorlage für die Behandlung vieler Patienten dient, ein unvorhergesehenes Ereignis aber eine Ausnahme ist, die nur einen Patienten betrifft. Würde die Workflow-Definition bei einem solchen Ausnahmeereignis geändert werden, wäre sie damit für die Mehrheit der Patienten nicht mehr passend.

Sollte ein Ausnahmeereignis so häufig eintreten, dass angenommen werden kann, dass die Mehrheit der Patienten davon betroffen ist, muss eine sogenannte Workflow-Schema-Evolution durchgeführt werden, d. h. die eigentliche Workflow-Definition muss verändert werden. Das ist aber nicht Aufgabe des Adaptations-Agenten und wird deshalb in dieser Arbeit nicht weiter behandelt (siehe [KG99] als weiterführende Referenz).

Ziel der vorliegenden Arbeit ist es, die Konzepte und Algorithmen für die beiden Hauptaufgaben des Agenten (Bestimmung der Adaptations-Region und Durchführung der Änderungen an einer betroffenen Workflow-Instanz) zu entwickeln und in einer prototypischen Implementierung umzusetzen.

1.4 Bisherige Ansätze

In diesem Abschnitt werden einige bestehende Workflow-Systeme untersucht, ob sie in der Lage sind, nach dem Auftreten eines logischen Fehlers eine Workflow-Instanz zur Laufzeit zu adaptieren.

Für die Adaptation eines Workflows zur Laufzeit muss die Workflow-Engine über Mechanismen zur *Forward-Recovery* verfügen, d. h. sie muss den Workflow unterbrechen und an der gleichen Stelle später wieder fortsetzen können. Wurde der Workflow unterbrochen, während sich gerade eine Aktivität in der Ausführung befand, wird der Workflow vor dem Wiederanlauf bis zum Beginn dieser Aktivität zurückgesetzt.
Kommerzielle Workflow-Management-Systeme wie z. B. IBM MQSERIES WORKFLOW bieten bisher nur *Backward-Recovery* an, das häufig nur die interne Datenbank umfasst und meist auch keine 2-Phasen-Commit-Protokolle verwendet.
Ein unterbrochener Workflow wird bei reinem Backward-Recovery bis zum Anfang zurückgesetzt. Eine andere Variante des Backward-Recovery ist die Anwendung *semantischer Check-Points*. Dabei wird nach einer Unterbrechung nicht bis zum Anfang sondern bis zu einem definierten Punkt zurückgesetzt, an dem es von der Semantik des Workflow her sinnvoll ist, bei einem Wiederanlauf zu beginnen.
Wenn ein logischer Fehler auftritt und der Workflow adaptiert werden muss, gelten diese Änderungen immer für den noch auszuführenden und nicht für den schon abgearbeiteten Teil des Workflows. Würde der Workflow zu einem festen Punkt zurückgesetzt werden, würden nach dem Wiederanlauf unter Umständen Aktivitäten erneut ausgeführt werden. Das ist in den wenigsten Anwendungsgebieten akzeptabel. In der Medizin würde das bedeuten, dass ein Patient zum Beispiel ein Medikament mehrfach bekommt, was unabsehbare Folgen haben kann.
Die Konsequenz wäre, dass bei logischen Fehlern der Workflow immer vollständig abgebrochen wird, um eine mehrfache Durchführung von Aktivitäten zu verhindern. Das wäre aber für den Anwender (z. B. den Arzt) völlig unakzeptabel, da nur ein Teil seiner Workflows vollständig abgearbeitet werden würde und der Rest an beliebigen Stellen abgebrochen wird.

Um logische Fehler sinnvoll zu behandeln, muss ein unterbrochener Workflow auch adaptiert werden können. Diese Möglichkeit bieten kommerzielle Produkte ebenfalls nicht. Selbst wenn

ein System über Mechanismen zur Forward-Recovery verfügen würde, könnte es kaum sinnvoll eingesetzt werden, da keine Änderungen am betroffenen Workflow vorgenommen werden können.
Für ein Workflow-System, das logische Fehler automatisch behandeln soll, sind also kommerzielle Workflow-Management-Systeme in ihrer ursprünglichen Form nicht geeignet.
Eine Lösung wäre, diese mit Modulen zu erweitern, die die Forward-Recovery und die Adaptation des unterbrochenen Workflows realisieren. Um solche Module implementieren zu können, müssten die Schnittstellen der Workflow-Engine bekannt sein. Da die Hersteller aber technische Details nur ungern veröffentlichen wollen, sind sie nur selten dazu bereit, die Schnittstellen-Definitionen zur Verfügung zu stellen.

Für die Adaptation von Workflows wurden in der Forschung einige Systeme entwickelt, zum Beispiel ADEPT$_{FLEX}$ [RD98]. Dieses System bietet eine Menge von Operatoren, mit deren Hilfe ein Workflow zur Laufzeit so umgebaut werden kann, dass wieder ein korrekter Workflow entsteht. Allerdings muss der Nutzer spezifizieren, welche Operatoren in welcher Reihenfolge angewendet werden sollen, und ihre Anwendung durchführen. Ein Workflow kann also nicht automatisch, d. h. ohne Nutzerinteraktion, verändert werden.
Andere Ansätze wie MOBILE [HH+99] oder WASA [VW98] sind ähnlich angelegt.

Um die Behandlung logischer Fehler für das Klinikpersonal weitgehend transparent zu halten, wäre es vorteilhaft, wenn man die Behandlung automatisieren könnte. Ansätze dazu findet man in WIDE [Ca98], das mit ECA-Regeln (Event-Condition-Action-Regeln) arbeitet. Temporale Aspekte, wie die zeitliche Auswirkung eines logischen Fehlers, werden aber nicht unterstützt.
Die KI-Planung beschäftigt sich damit, wie Pläne, die durch eine Veränderung der Situation nicht mehr adäquat sind, automatisch an die neue Situation angepasst werden können. Der Schwerpunkt liegt hier auf der Automatisierung der Anpassung, die ohne Eingriff von außen erfolgen soll. Arbeiten dazu sind z. B. [Ham90]. Da dort Datenfluss-Aspekte aber nur unzureichend unterstützt werden, sind solche Ansätze für den Einsatz im Workflow-Management nur bedingt geeignet.

1.5 Gliederung

Die Arbeit ist wie folgt gegliedert:
In Kapitel 2 wird zuerst erläutert, wie logische Fehler behandelt werden können. Danach werden die Architektur und das Meta-Modell des Workflow-Management-Systems AGENTWORK vorgestellt.
Anschließend wird in Kapitel 3 ein Überblick über den Adaptations-Agenten gegeben und es werden verschiedene Adaptations-Strategien vorgestellt.
Die Konzepte für die Realisierung der beiden Hauptaufgaben des Agenten werden in den Kapiteln 4 (Bestimmung der Adaptations-Region) und 5 (Durchführung der Adaptationen) erläutert.
Kapitel 6 beschreibt die Ziele und die Umsetzung der prototypischen Implementierung.
Den Abschluss bilden eine Zusammenfassung der Ergebnisse und ein Ausblick auf die weitere Entwicklung (Kapitel 7).

2. Das Workflow-System AGENTWORK

In diesem Kapitel wird zunächst dargestellt, wie logische Fehler behandelt werden können. Anschließend werden die Architektur und das Workflow-Meta-Modell des Systems AGENTWORK vorgestellt.

2.1 Behandlung logischer Fehler

Nach dem Auftreten eines logischen Fehlers dürfen bestimmte Aktivitäten eines Workflows nicht mehr ausgeführt werden, müssen verzögert oder zusätzlich durchgeführt werden. Der Kontrollfluss des Workflows muss also entsprechend verändert werden.
Welche Aktionen an einem betroffenen Workflow durchzuführen sind, wird über Kontroll-Aktionen angegeben (vgl. Abschnitt 1.3). Eine Kontroll-Aktion sieht beispielsweise so aus (bezogen auf das Beispiel aus Abschnitt 1.3):

```
drop(Verabreiche Vincristin)@14,day
```

Es sind also alle Knoten zu löschen, denen der Aktivitätstyp *Verabreiche Vincristin* zugeordnet ist und die innerhalb der nächsten 14 Tage zur Ausführung kommen werden. Der @-Operator dient zur syntaktischen Trennung der eigentlichen Kontroll-Aktion von ihrem Gültigkeitsintervall.
Das Gültigkeitsintervall der Kontroll-Aktion (hier 14 Tage) bestimmt die Adaptations-Region, also die Teile der betroffenen Workflows, die während des Gültigkeitsintervalls voraussichtlich ausgeführt werden (vgl. Abschnitt 1.3).
Tritt ein logischer Fehler auf, werden eine oder mehrere Kontroll-Aktionen gefolgert, die beschreiben, wie der Fehler zu behandeln ist.

Dabei werden zwei Klassen von logischen Fehlern unterschieden:

- *globale* logische Fehler: Sie betreffen den gesamten Workflow, der nicht mehr weiter ausgeführt werden darf.
- *lokale* logische Fehler: Sie betreffen nur einige Aktivitäten eines Workflows. Dieser kann nach Änderungen am Kontrollfluss weiter ausgeführt werden.

Die oben beschriebenen Kontroll-Aktionen werden beim Auftreten lokaler logischer Fehler mit Hilfe von Regeln gefolgert. Eine solche Regel kann umgangssprachlich formuliert z. B. so lauten:

> *Wenn der Wert x > 5000 ist, ersetze für die nächsten drei Tage das Medikament A durch das Medikament B.*

Die zweite Satzhälfte gibt an, was zu tun ist, und entspricht der gefolgerten Kontroll-Aktion. Der Aktivitätstyp ist *Verabreiche Medikament A*, die durchzuführende Aktion ist *Ersetze A durch B* und das Gültigkeitsintervall ist *drei Tage*. Der Adaptations-Agent muss jetzt zunächst den Teil des Workflows bestimmen, der in den nächsten drei Tagen voraussichtlich durchlaufen werden wird, und dann darin alle Aktivitäten, die das Medikament A verabreichen durch Aktivitäten ersetzen, die das Medikament B verabreichen.

Bei globalen logischen Fehlern werden keine Adaptationen durchgeführt. Es sind aber eventuell bereits durchgeführte Aktivitäten zu kompensieren. So kann z. B. eine bereits verschickte Mahnung durch das Auftreten eines globalen logischen Fehlers gegenstandslos geworden sein und muss durch ein Entschuldigungsschreiben zurückgenommen werden. Solche Kompensationsoperationen können nicht im betroffenen Workflow eingebaut werden, da dieser nicht mehr weiter ausgeführt werden darf. Sie können entweder in einen passenden, parallel ausgeführten Workflow eingefügt werden, was dann einer Adaptation dieses Workflows entspricht, oder es wird ein eigener Workflow für die Kompensationsoperationen erzeugt und gestartet, nachdem der Workflow, der den globalen logischen Fehler verursacht hat, abgebrochen wurde.

Allgemein umfasst die Behandlung logischer Fehler drei Schritte: die Erkennung eines Fehlers, den Abbruch bzw. die Unterbrechung eines Workflows und die Kompensation oder Adaptation.

- *Erkennung eines Fehlers:*
  Dafür können z. B. Bedingungs-Ereignis- oder andere logische Regeln formuliert werden, die festlegen, was zu tun ist, wenn bestimmte Ereignisse eintreten. Feuert eine solche Regel, wird der nächste Schritt in der Behandlung logischer Fehler, der Abbruch oder die Unterbrechung eines Workflows, veranlasst und es werden eine oder mehrere lokale Kontroll-Aktionen gefolgert, die die Adaptations- oder Kompensationsoperationen für den Workflow spezifizieren.

- *Abbruch / Unterbrechung des Workflows*:
  Bei einem globalen logischen Fehler wird der Workflow vollständig abgebrochen. Ist ein lokaler logischer Fehler eingetreten, wird der Workflow nur unterbrochen, damit die Änderungen durchgeführt werden können.
  Es wäre auch denkbar, die Adaptationen vorzunehmen, während der Workflow weiter ausgeführt wird. Da aber die Dauer der Adaptation nicht vernachlässigt werden kann, da z. B. Interaktion mit einem Benutzer nötig ist, würden möglicherweise Teile des Workflows geändert, die in der geänderten Form gar nicht mehr ausgeführt werden, da der Kontrollfluss bereits weiter ist. Um solche Situationen zu vermeiden, wird der Workflow vor Durchführung der Adaptationen unterbrochen und erst nach erfolgreicher Änderung weiter ausgeführt.

- *Kompensation /Adaptation:*
  Als letzter Schritt der Behandlung eines logischen Fehlers sind die Kompensationen bzw. Adaptationen durchzuführen. Was dabei genau zu tun ist, wird durch die gefolgerten Kontroll-Aktionen bestimmt.

Nachdem diese drei Schritte durchgeführt wurden, können die unterbrochenen Workflows weiter ausgeführt und eventuell neu generierte Workflows gestartet werden.

Das Workflow-System soll die Anwender, z. B. das Klinikpersonal oder die Mitarbeiter eines Reisebüros, bei der Arbeit unterstützen und soweit wie möglich von bestimmten Aufgaben entlasten. Wenn bei jedem Auftreten eines logischen Fehlers die oben aufgeführten Schritte von einem menschlichen Benutzer in Interaktion mit dem System durchgeführt werden müssten, würde das den Nutzer sehr viel Zeit kosten, die ihm dann für seine anderen Aufgaben fehlt. Deshalb wäre es wünschenswert, dass die Erkennung und Behandlung von logischen Fehlern weitgehend automatisch ohne Eingreifen eines Nutzers erfolgt. Eine Interaktion sollte nur in Ausnahmefällen oder zur Bestätigung und Rückversicherung notwendig sein.
Im Folgenden wird skizziert, wie die drei Schritte zur Behandlung eines logischen Fehlers zumindest teilweise automatisiert werden können.

- Zur *Erkennung* von logischen Fehlern kann man Ereignisse, die im System eintreten, wie z. B. der Eintrag neuer Laborwerte in eine Datenbank mit Hilfe von logischen Regeln daraufhin überwachen, ob sie ein Anwendungsereignis auslösen. Wenn das der Fall ist, feuern eine oder mehrere der Regeln und es werden wie oben beschrieben ein Abbruch oder eine Unterbrechung veranlasst und verschiedene lokale Kontroll-Aktionen gefolgert.

- Der *Abbruch* bzw. die *Unterbrechung* eines Workflows kann von der Workflow-Engine automatisch durchgeführt werden, wenn diese über die entsprechenden Mechanismen verfügt.

- Die *Adaptation* der betroffenen Workflows kann vom Adaptations-Agenten mit Hilfe der gefolgerten Kontroll-Aktionen automatisch vorgenommen werden. Nur bei Zweideutigkeiten wird der Nutzer in die Änderungen einbezogen. Da möglicherweise zusätzliche Adaptationen notwendig sind, die nicht automatisch gefolgert werden konnten, muss der Nutzer vor der weiteren Ausführung des Workflows entscheiden, ob die Änderungen korrekt sind oder ob er sie nochmals überarbeiten möchte.

2.2 Architektur des Workflow-Systems AGENTWORK

Im Laufe des Projektes HEMATOWORK wurde deutlich, dass die Möglichkeit der Adaptation von Workflows zur Laufzeit eine wichtige Anforderung an ein geeignetes System für die Unterstützung der studienorientierten Hämato-Onkologie darstellt. Durch unerwartete Nebenwirkungen oder Infektionen wird es immer wieder nötig, die Behandlungspläne zu verändern. Das Auftreten eines solchen Anwendungsereignisses löst aus Sicht des Workflow-Systems einen logischen Fehler aus.

Da das medizinische Personal durch den Einsatz eines Workflow-Systems in der Planung und Durchführung der Therapie unterstützt werden soll und die Überwachung der Ereignisse und die Änderung der Workflows sehr zeitaufwändig sind, müssen die Erkennung von Fehlern und die Adaptationen teilautomatisch erfolgen. Nur wenn z. B. nicht eindeutig ist, wie auf einen Fehler reagiert werden soll, sollte ein Nutzer hinzugezogen werden. Eine weitere Nutzerinteraktion ist vor der weiteren Ausführung einer adaptierten Workflow-Instanz erforderlich, um dem Nutzer die Möglichkeit zu geben, die vorgenommenen Änderungen gegebenenfalls zu überarbeiten, und sie dann abschließend zu bestätigen.

Da die verfügbaren Systeme aus den in Abschnitt 1.4 genannten Gründen hier nur eine unzureichende Unterstützung bieten, wurde das Workflow-Management-System AGENTWORK entwickelt, das Ereignisse überwacht, Kontroll-Aktionen folgert und diese ausführt. Das System wurde dabei unabhängig von der konkreten Anwendung des Projektes HEMATOWORK entwickelt und ist so auch für andere Anwendungsgebiete einsetzbar.

Die einzelnen Schichten des Systems sind in Abbildung 2-1 dargestellt und sollen im Folgenden kurz erläutert werden.

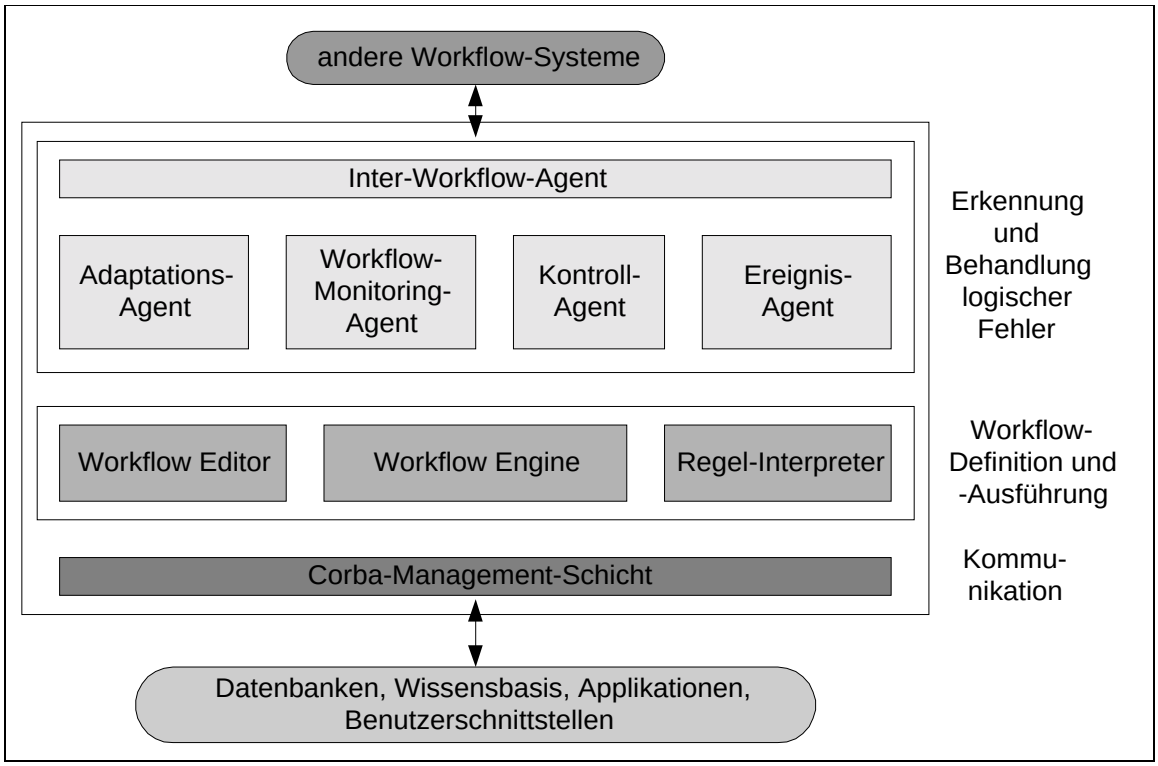


Abb. 2-1: Architektur des Workflow-Systems AGENTWORK [Grafik aus Projektbestand].

Die Architektur gliedert sich in drei Schichten: die Schicht für Workflow-Definition und -Ausführung, die Schicht zur Erkennung und Behandlung logischer Fehler und die Kommunikations-Schicht:

- *Schicht für Workflow-Definition und -Ausführung:*
  Sie besteht aus dem *Workflow-Editor*, der *Workflow-Engine* und einem *Regel-Interpreter.* Im Workflow-Editor werden die Workflows über eine grafische Oberfläche definiert. Die Workflow-Engine führt Workflows aus und ist in der Lage, diese an einer beliebigen Stelle zu unterbrechen und später fortzusetzen.
  Der Regel-Interpreter wird zum Einen von der Engine benötigt, um Verzweigungsbedingungen auswerten zu können, zum Anderen benötigen ihn die Agenten zur Erkennung und Behandlung von logischen Fehlern.

- *Schicht zur Erkennung und Behandlung logischer Fehler:*

  Sie setzt sich aus fünf Agenten (*Ereignis-Agent*, *Kontroll-Agent*, *Adaptations-Agent*, *Workflow-Monitoring-Agent*, *Inter-Workflow-Agent*) zusammen, die gemeinsam die Erkennung und Behandlung logischer Fehler realisieren. Die Aufgaben der einzelnen Agenten und ihre Zusammenarbeit werden im Folgenden skizziert.

  Der Ereignis-Agent überwacht mit Hilfe von Regeln alle Ereignisse im System (z. B. den Eintrag neuer Laborwerte in eine Datenbank). Feuern eine oder mehrere Regeln, d. h. ihre Bedingung werden zu wahr evaluiert, folgert er entsprechende globale Kontroll-Aktionen, um die betroffenen Workflow-Instanzen abzubrechen oder zu suspendieren, und lokale Kontroll-Aktionen, um den Kontrollfluss zu ändern. Anschließend informiert er den Kontroll-Agenten darüber. Dieser veranlasst daraufhin die Workflow-Engine, die betroffenen Instanzen vollständig abzubrechen oder zu unterbrechen.

  Wurde die Instanz erfolgreich unterbrochen, ruft der Kontroll-Agent den Adaptations-Agenten auf, der die Adaptationen entsprechend den gefolgerten lokalen Kontroll-Aktionen vornimmt. Anschließend benachrichtigt er den Kontroll-Agenten und die Workflow-Engine, die daraufhin die Ausführung der Instanz fortsetzt.

  Falls der Adaptations-Agent die Adaptations-Region nicht korrekt bestimmt hat, müssen eventuell Änderungen rückgängig gemacht oder nachgeholt werden. Um das feststellen zu können, muss die weitere Ausführung einer geänderten Instanz überwacht werden. Diese Überwachung ist die Aufgabe des Workflow-Monitoring-Agenten.

  Wenn Abhängigkeiten zwischen unterschiedlichen Workflows bestehen, können sich Änderungen an einer Instanz auch auf andere Workflows auswirken. Der Inter-Workflow-Agent überwacht solche Abhängigkeiten zwischen Workflows (siehe [MR00] als weiterführende Referenz).

  Die Komponenten dieser Schicht haben ein „intelligentes“ Verhalten und bearbeiten ihre Teilaufgaben selbständig. Um das Gesamtproblem der dynamischen Workflow-Adaptation zu lösen, müssen sie aber zusammenarbeiten und auf Änderungen in ihrer Umgebung reagieren. Da diese Merkmale (Intelligenz, Autonomie, Kooperation, Reaktion auf die Umgebung) dem Paradigma der Agenten-basierten Modellierung ([$WJ95_A$], [FG97]) entsprechen, werden die Komponenten als Agenten bezeichnet. Die vorliegende Arbeit beschäftigt sich aber nicht mit Agentenforschung, sondern benutzt die Bezeichnung

lediglich, um die oben genannten Eigenschaften der Komponenten zu verdeutlichen. (Literatur zur Agentenforschung sind z. B. [MSR99, MS99, BEM99].)

- *Kommunikations-Schicht:*
  Die einzelnen Komponenten des Systems müssen miteinander und mit externen Datenbanken, Wissensbasen oder Applikationen kommunizieren. Da die Schnittstellen sehr unterschiedlich sein können, wird eine Kommunikations-Schicht eingeführt. Diese abstrahiert von den technischen Details der einzelnen Komponenten und stellt durch die Verwendung von CORBA [OMG97] eine einheitliche Schnittstelle für die Kommunikation zur Verfügung. Damit können z. B. auch heterogene Datenbanken in das System integriert werden.

Die Konzeption des Gesamtsystems wird in [Mue00] beschrieben. Die einzelnen Komponenten werden zum Teil in Diplomarbeiten realisiert. So ist es Ziel der vorliegenden Arbeit, den Adaptations-Agenten zu entwickeln. Auch die Workflow-Engine [Die00] und der Workflow-Editor [Boe00] sind Gegenstand zweier Diplomarbeiten.

2.3 Workflow-Meta-Modell des Systems AGENTWORK

Das Workflow-Meta-Modell enthält alle Konstrukte, mit denen Kontroll- und Datenfluss beschrieben werden. Für den Kontrollfluss stehen z. B. verschiedene Arten von Knoten und Transitionen und unterschiedliche Aktivitätstypen zur Verfügung. Der Datenfluss enthält beispielsweise die ein- und ausgehenden Objekte einer Aktivität.
Im Folgenden werden die wichtigsten Konzepte für die Workflow-Definition und für die Workflow-Ausführung vorgestellt.

2.3.1 Elemente der Workflow-Definition

Es gibt drei Typen von Knoten: *Aktivitätsknoten, Kontrollknoten* und *Kommunikationsknoten*.

- *Aktivitätsknoten* repräsentieren die einzelnen Aktivitäten des Workflows, wobei zwei Arten unterschieden werden. Eine *Basis-Aktivität* repräsentiert eine einzelne, aus Sicht des Betrachters atomare Aktivität, wie z. B. die Verabreichung eines Medikaments.
  Treten in einem Workflow eine Menge von Aktivitäten, beispielsweise mehrere Untersuchungen, nacheinander auf, die in der gleichen Reihenfolge in einem eigenen Workflow zusammengefasst sind, muss man im übergeordneten Workflow die einzelnen Aktivitäten nicht explizit einfügen. Stattdessen wird eine *Komplex-Aktivität* eingefügt, die den entsprechenden Subworkflow, der die Aktivitäten enthält, repräsentiert.

- *Kontrollknoten* dienen zur Modellierung der Ablauflogik des Workflows. So leiten Splitknoten z. B. die parallele Ausführung mehrerer Pfade ein oder spezifizieren konditionale Verzweigungen. Loop-Start- und Loop-End-Knoten ermöglichen beispielsweise die wiederholte Ausführung einer bestimmten Menge von Aktivitäten.

- *Kommunikationsknoten* kennzeichnen die Stellen im Workflow, an denen eine Kommunikation mit einem anderen Workflow erfolgen soll. Man unterscheidet hier Kommunikationsknoten für die eingehende Kommunikation und solche für die ausgehende Kommunikation.

Bei den Transitionen unterscheidet man *Transitionen ohne Bedingung, konditionale Transitionen, temporale Transitionen*, *Synchronisations-Transitionen* und *Objektfluss-Transitionen*.

- *Transition ohne Bedingung* werden dann verwendet, wenn es keine Bedingung für den Übergang vom Quell- zum Zielknoten gibt.

- Bei *konditionalen Transitionen* wird der nächste Knoten nur ausgeführt, wenn die angegebene Bedingung erfüllt ist. Dieser Transitionstyp kommt nur nach Split-Knoten und zwischen Loop-End- und Loop-Start-Knoten vor. Um die Bedingung auswerten zu kön-

nen, sind häufig Daten erforderlich. Diese werden als Eingabedaten der konditionalen Transition bezeichnet.

In Abbildung 2-2 ist eine Verzweigung mit zwei konditionalen Transitionen abgebildet. Wenn der Wert x kleiner als 5000 ist, wird als nächstes die Aktivität A ausgeführt, ist x größer oder gleich 5000, wird die Aktivität B ausgeführt.

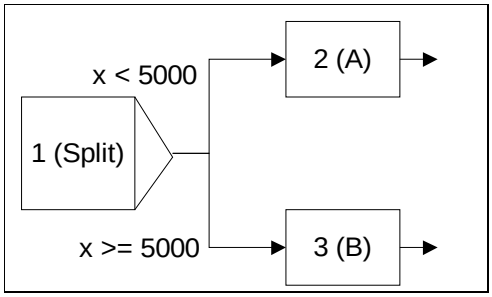


Abb. 2-2: Konditionale Transitionen.

- *Temporale Transitionen* haben die Semantik, dass nach Beendigung des Quellknotens bis zur Ausführung des Zielknotens eine bestimmte Zeit gewartet werden muss.

  So besagt z. B. eine Kante zwischen den Aktivitäten A und B, an der ein Zeitwert *3 Stunden* steht, dass nach Beendigung der Aktivität A drei Stunden gewartet werden muss, bevor die Aktivität B gestartet werden darf (s. Abbildung 2-3).

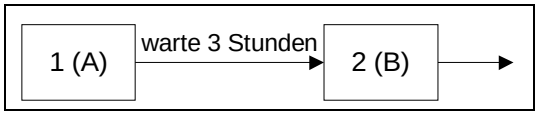


Abb. 2-3: Temporale Transition.

- *Synchronisations-Transitionen* werden verwendet, um die Reihenfolge zweier Aktivitäten, die in parallelen Pfaden liegen, festzulegen. Das ist z. B. nötig, um sicherzustellen, dass Medikamente in einer bestimmten Reihenfolge verabreicht werden.

  Angenommen durch Aktivität A wird Medikament 1 gegeben und durch Aktivität F Medikament 2, das immer nach Medikament 1 gegeben werden muss. Liegen die beiden Aktivitäten wie in Abbildung 2-4 dargestellt in zwei parallelen Pfaden, muss die richtige

Reihenfolge der Verabreichung durch eine Synchronisations-Transition sichergestellt werden. Im Beispiel führt diese von Knoten 2 (Aktivität A) zu Knoten 7 (Aktivität F) und ist gestrichelt gezeichnet.

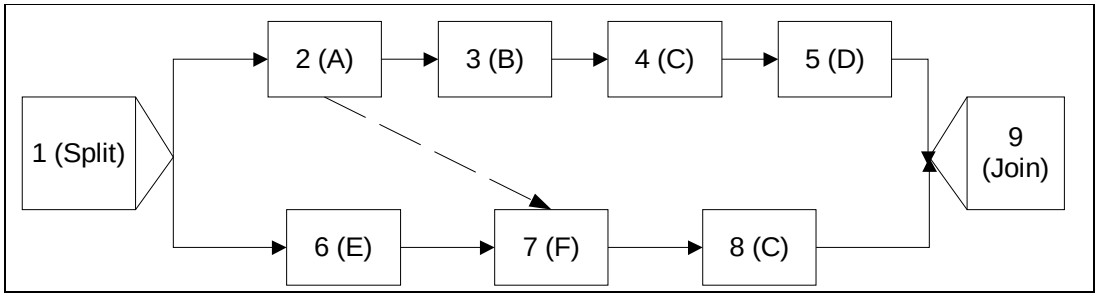


Abb. 2-4: Synchronisations-Transition.

- *Objektfluss-Transitionen* modellieren den Weg von Daten zwischen zwei Knoten.

Es gibt mehrere Arten von logischen Regionen: *Sequenzen*, verschiedene Arten von *Verzweigungen* und *Schleifen*. In den folgenden Abbildungen werden die verschiedenen Regionen dargestellt und erläutert. Die Knoten sind durchnummeriert, in Klammern ist der Typ angegeben. Dabei bezeichnen Großbuchstaben einen Aktivitätstyp, bei Kommunikationsknoten steht Comm-In für einen Knoten, bei dem Daten von einem anderen Workflow eingehen, und Comm-Out für einen Knoten, bei dem Daten an einen anderen Workflow weggeschickt werden. Bei Kontrollknoten ist der Typ angegeben.

Jeder Workflow besteht mindestens aus einer *Start-Ende-Region*. Diese beginnt mit einem Startknoten, der den Beginn eines Workflows kennzeichnet, und endet mit einem Endeknoten, der das Ende eines Workflows anzeigt. Jeder Workflow darf nur eine Start-Ende-Region enthalten.

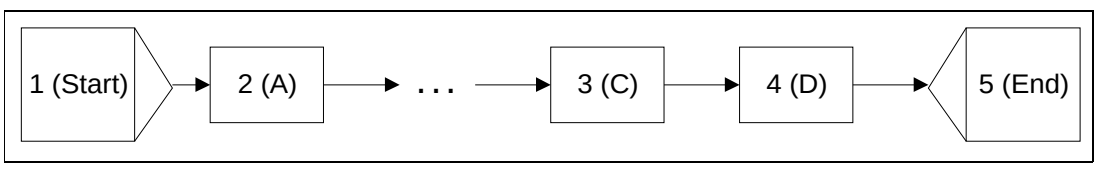


Abb.2-5: Start-Ende-Region.

In einer *Sequenz* werden die Knoten nacheinander abgearbeitet, d. h. wenn die Aktivität A zu Ende ist, wird die Aktivität B gestartet, usw. Eine Sequenz kann nur Aktivitäts- und Kommunikationsknoten enthalten.

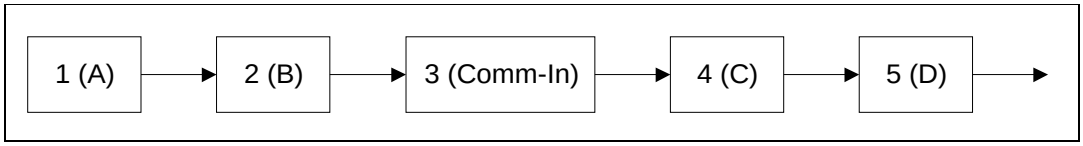


Abb.2-6: Sequenz.

Bei einer *AND-Verzweigung* werden alle Pfade gleichzeitig gestartet und parallel abgearbeitet. Eine AND-Verzweigung beginnt mit einem AND-Split-Knoten und endet mit einem Join-Knoten. Die gesamte Region vom Split- bis zum Join-Knoten wird auch als *Split/Join-Region* bezeichnet.

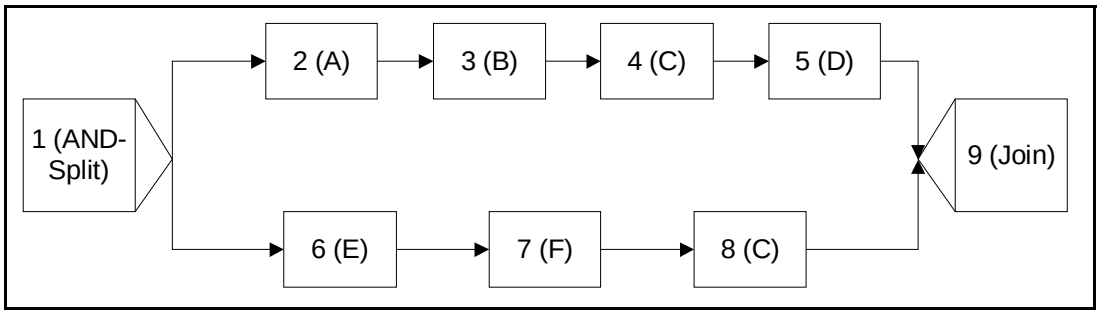


Abb. 2-7: AND-Verzweigung.

Bei einer *OR-Verzweigung* werden nur die Pfade ausgeführt, deren eingehende Transitions-Bedingung erfüllt ist. OR-Verzweigungen beginnen mit einem OR-Split-Knoten und enden mit einem Join-Knoten. Die gesamte Region wird ebenfalls als Split/Join-Region bezeichnet.

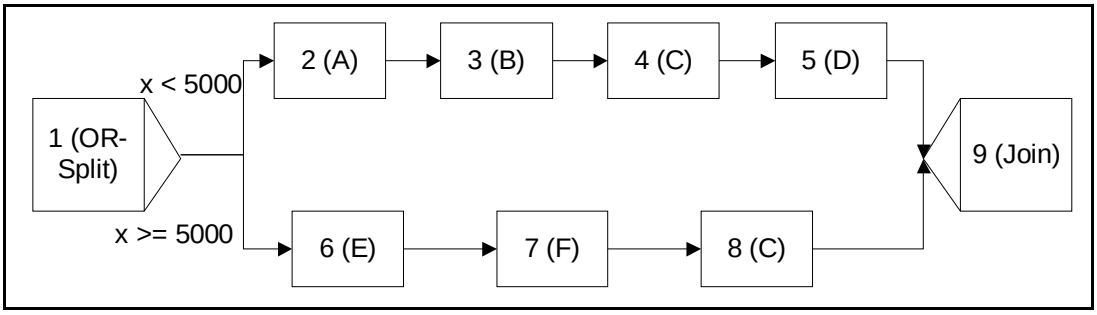


Abb. 2-8: OR-Verzweigung.

Es gibt drei unterschiedliche Typen von Join-Knoten, die in der Tabelle 2-1 zusammengefasst sind.

Join-Typ	Erläuterung
ALL-Join	Kontrollfluss geht zu nächstem Knoten nach Join-Knoten über, wenn alle Pfade Join-Knoten erreicht haben.
One-Join-Complete	Kontrollfluss geht zu nächstem Knoten nach Join-Knoten über, wenn der erste Pfad den Join-Knoten erreicht hat. Die übrigen Pfade werden beendet.
One-Join-Cancel	Kontrollfluss geht zu nächstem Knoten nach Join-Knoten über, wenn der erste Pfad den Join-Knoten erreicht hat. Die Ausführung der übrigen Pfade wird abgebrochen.

Tabelle 2-1: Join-Typen.

Ein ALL-Join-Knoten modelliert die übliche Semantik eines Join-Knotens, dass im Kontrollfluss erst weitergegangen wird, wenn alle Verzweigungspfade den Join-Knoten erreicht haben. Er kann sowohl in Verbindung mit einem AND-Split- als auch mit einem OR-Split-Knoten verwendet werden. Eine OR-Verzweigung darf nur mit einem ALL-Join-Knoten beendet werden, da zur Modellierungszeit nicht bekannt ist, welche Pfade zur Laufzeit tatsächlich ausgewählt werden und ob die Daten des ersten Pfades, der den Join-Knoten erreicht, dann ausreichen, um den Kontrollfluss zum nächsten Knoten nach dem Join-Knoten übergehen zu lassen.

Ein One-Join-Complete-Knoten hat folgende Semantik: Wenn z. B. in einer AND-Verzweigung mehrere Untersuchungen vorgenommen werden und für die weitere Ausführung des Workflows nach der Split/Join-Region vorerst die Diagnose von einer Untersuchung ausreicht, kann der Kontrollfluss zum Knoten nach dem Join-Knoten übergehen, wenn der erste Pfad diesen erreicht hat. Falls die Daten, die durch die Untersuchungen der übrigen Pfade entstehen, aber später noch benötigt werden, werden die restlichen Split-Pfade auch zu Ende geführt, damit die Daten nicht später neu erzeugt werden müssen.
Bei einem One-Join-Cancel-Knoten ist nur entscheidend, dass ein Pfad den Join-Knoten erreicht. Die Ergebnisse aus den übrigen Pfaden werden nicht mehr benötigt. Deshalb geht hier der Kontrollfluss zum Knoten nach dem Join-Knoten über, sobald der erste Pfad diesen erreicht hat. Die übrigen Pfade werden nach Beendigung der gerade ausgeführten Aktivität abgebrochen.

Bei einer *Schleife* wird der Schleifenkörper (in der Abb. 2-9 die Knoten 2, 3, 4 und 5) solange durchlaufen, bis die Schleifenbedingung nicht mehr erfüllt ist. Da die Auswertung erst am Ende eines Durchlaufs erfolgt, wird die Schleife mindestens einmal durchlaufen.

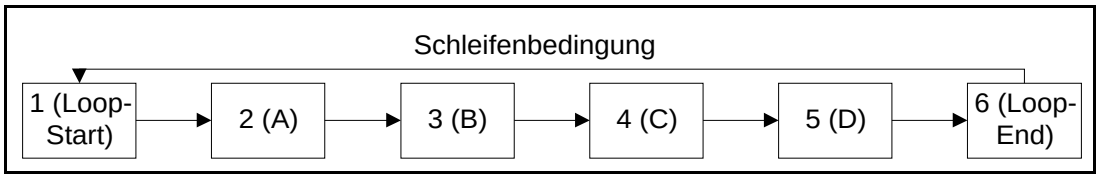


Abb. 2-9: Schleife.

Diese Regionen von logisch zusammenhängenden Aktivitäten, die von Kontrollknoten begrenzt sind, werden auch als Kontroll-Regionen bezeichnet. Alle Kontroll-Regionen außer der Start-Ende-Region können bei der Definition eines Workflows beliebig geschachtelt werden. Dabei darf eine Region aber nicht innerhalb einer anderen Region beginnen und außerhalb dieser enden. Die Anordnung von Knoten in der Reihenfolge „Loop-Start - Split - Loop-End - Join“ ist also beispielsweise nicht möglich. Die Start-Ende-Region muss immer die äußerste Region bilden.

Der Datenfluss modelliert den Fluss von Daten zwischen Knoten, zwischen Knoten und Datenbanken und zu konditionalen Transitionen. Der Datenfluss innerhalb des Workflows wird als *interner* Datenfluss, der Datenfluss zwischen Knoten oder Transitionen und Datenbanken als *externer* Datenfluss bezeichnet. In Abbildung 2-10 gehören die Daten D1 zum internen, die Daten D2 zum externen Datenfluss.

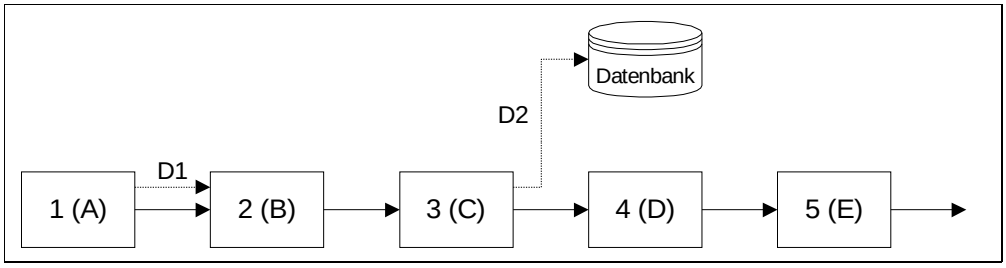


Abb. 2-10: Datenfluss.

2.3.2 Workflow-Ausführungsmodell

Während der Ausführung eines Workflows nehmen die Knoten und Transitionen verschiedene Zustände an. Diese müssen auch bei der Bestimmung der Adaptations-Region berücksichtigt werden.

In den folgenden beiden Abbildungen 2-11 und 2-12 werden die Knoten- und Transitions-Zustände und die möglichen Übergänge zwischen ihnen dargestellt.

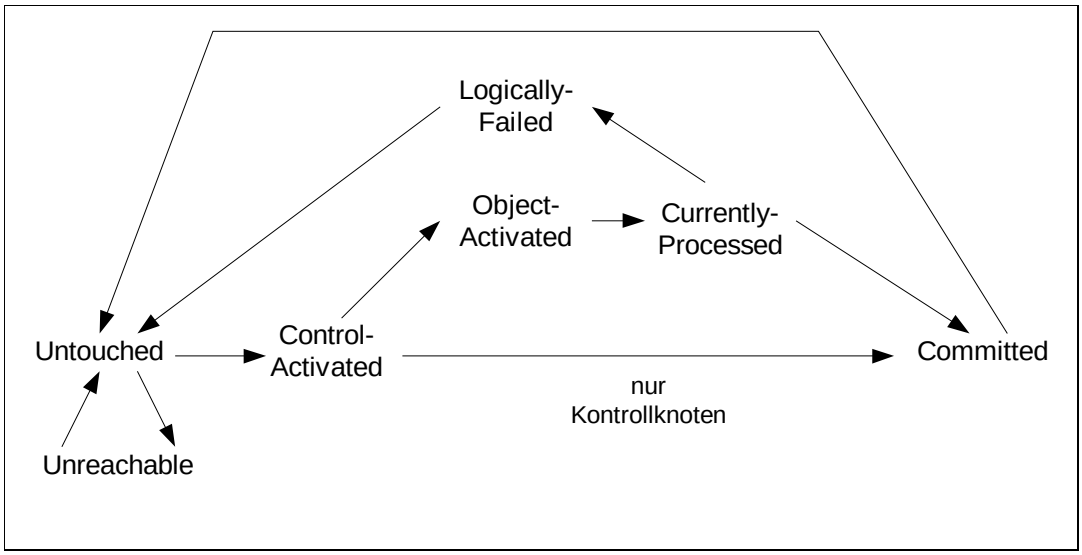


Abb. 2-11: Knotenzustände und Zustandsübergänge.
(Auch OR-Split-Knoten gehen vom Zustand Control-Activated direkt in den Zustand Committed über, da die Auswertung der Verzweigungsbedingungen erst an den vom OR-Split-Knoten ausgehenden konditionalen Transitionen erfolgt und die benötigten Daten erst dort zur Verfügung stehen müssen.)

Zu Beginn der Ausführung eines Workflows sind alle Knoten im Zustand Untouched, wurden also noch nicht vom Kontrollfluss erreicht.

Erreicht der Kontrollfluss einen Knoten, geht dieser in den Zustand Control-Activated über. Handelt es sich um einen Kontrollknoten, ist der nächste Zustand Committed, da diesem Knoten keine Aktivitäten zugeordnet sind, die durchgeführt werden müssen. Auch OR-Split-Knoten gehen direkt vom Zustand Control-Activated in den Zustand Committed über, da die Bedingungsauswertung über die ausgehenden konditionalen Transitionen des Split-Knotens erfolgt. Die Daten, die zur Bedingungsauswertung benötigt werden, sind also den konditionalen Transitionen und nicht den OR-Split-Knoten zugeordnet.

Bei Aktivitäts- und Kommunikationsknoten wird als nächstes untersucht, ob alle benötigten Eingabedaten vorhanden sind. Ist das der Fall, erreichen sie den Zustand Object-Activated. Danach werden die Aktivitäten bzw. die Kommunikation ausgeführt, der Knoten befindet sich im Zustand Currently-Processed. Verläuft die Ausführung erfolgreich, ist der nächste und abschließende Zustand Committed. Tritt während der Ausführung ein logischer Fehler auf, geht der Knoten in den Zustand Logically-Failed über. Nach der erfolgreichen Unterbrechung und Adaptation des Workflows werden solche Knoten in den Zustand Untouched zurückgesetzt, damit sie erneut ausgeführt werden können.

Knoten die in einem OR-Split-Pfad liegen, der nicht zur Ausführung ausgewählt wird, gehen direkt vom Zustand Untouched in den Zustand Unreachable über, sie können also vom Kontrollfluss nicht mehr erreicht werden.
Wird nach einer Unterbrechung der Kontrollfluss um einige Knoten zurückgesetzt, können Knoten, die im Zustand Committed oder Unreachable waren, wieder in den Zustand Untouched übergehen, damit sie erneut ausgeführt werden können. Setzt der Kontrollfluss z. B. bis vor einen OR-Split-Knoten zurück, muss auch die Auswahl der Pfade rückgängig gemacht werden, also gehen alle Knoten der Split/Join-Region in den Zustand Untouched über. Wird aber nur bis zum Beginn der einzelnen Pfade der OR-Verzweigung zurückgesetzt, bleiben nicht ausgewählte Pfade im Status Unreachable, da die Bedingungsauswertung nicht rückgängig gemacht wird.

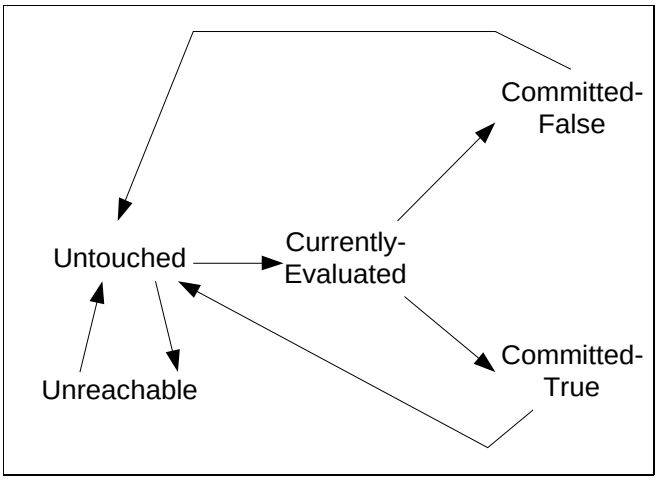


Abb. 2-12: Transitions-Zustände und Zustandsübergänge.

Alle Transitionen befinden sich vor der Ausführung eines Workflows im Zustand Untouched. Erreicht der Kontrollfluss eine Transition werden eventuell vorhandene Bedingungen ausgewertet, die Kante befindet sich im Zustand Currently-Evaluated. Abhängig vom Ergebnis der Auswertung ist der nächste Zustand entweder Committed-True oder Committed-False. Die Kanten in einem nichtausgewählten OR-Split-Pfad gehen wie die Knoten vom Zustand Untouched direkt in den Zustand Unreachable über. Wird der Kontrollfluss nach einer Unterbrechung zurückgesetzt, können die Transitionen in den Zuständen Committed-True, Committed-False und Unreachable analog zu den Knoten wieder in den Zustand Untouched übergehen.

3. Überblick über den Adaptations-Agenten

Der Adaptations-Agent ist dafür verantwortlich, strukturelle Änderungen an einem von einem logischen Fehler betroffenen Workflow vorzunehmen.
Dazu muss er wissen, welche Aktivitätstypen von einem logischen Fehler betroffen sind, welche Änderungen durchzuführen sind und auf welches Gültigkeitsintervall *T* sich diese beziehen. Diese Informationen werden in *Kontroll-Aktionen* zusammengefasst (vgl. Abschnitt 1.3).
Die eigentliche Durchführung der Adaptationen kann mit zwei unterschiedlichen Strategien erfolgen. Bei der *reaktiven Strategie* wird gewartet, bis eine von einem logischen Fehler betroffene Aktivität zur Ausführung kommt. Dann wird für diese Aktivität die Änderung vorgenommen (z. B. das Löschen eines Aktivitätsknotens). Hingegen wird bei der *prädiktiven Strategie* versucht, den Teil des noch auszuführenden Workflows zu bestimmen, der innerhalb des Gültigkeitsintervalls wahrscheinlich zur Ausführung kommen wird. Für alle von dem logischen Fehler betroffenen Knoten in diesem Teil des Workflows werden dann die Änderungen durchgeführt. Vor- und Nachteile der beiden Strategien werden später diskutiert.
Das Format, die Typen und Semantiken der Kontroll-Aktionen und die Adaptations-Strategien wurden in [Mue00] entwickelt und werden in diesem Kapitel zusammenfassend erläutert, da sie die Basis für die Arbeit des Adaptations-Agenten bilden.

3.1 Kontroll-Aktionen

Eine Kontroll-Aktion informiert den Adaptations-Agenten darüber, welche Aktivitätstypen einer Workflow-Instanz von einem logischen Fehler betroffen sind (z. B. die Verabreichung eines Medikaments), wie die Knoten zu behandeln sind (z. B. Entfernen oder Ersetzen) und für welches Gültigkeitsintervall *T* die Kontroll-Aktion gilt.

Es gibt sechs Typen von Kontroll-Aktionen: *drop*, *replace*, *add*, *add-repetitively*, *delay* und *check*. Im Folgenden werden deren Semantiken beschrieben und anschließend wird die Bedeutung des Gültigkeitsintervalls genauer erläutert.

- *drop(A)@T*:

  Aktivitäten vom Typ A dürfen im Intervall *T* nicht ausgeführt werden.

  Für den Adaptations-Agenten bedeutet das, dass er alle Knoten, denen der Aktivitätstyp A zugeordnet ist und die im Intervall *T* zur Ausführung kommen werden, aus der von dem logischen Fehler betroffenen Instanz löschen muss.

- *replace(A, B)@T*:

  Im Intervall *T* muss jede Aktivität vom Typ A durch eine Aktivität vom Typ B ersetzt werden.

- *add(A)@T*:

  Im Gültigkeitsintervall *T* muss eine Aktivität vom Typ A einmal zusätzlich zu den bereits im Workflow enthaltenen Aktivitäten ausgeführt werden.

  Der Adaptations-Agent muss zwischen den Knoten der Instanz, die innerhalb von *T* voraussichtlich ausgeführt werden, eine passende Stelle finden, an der ein Knoten eingefügt werden kann. Diesem wird dann der Aktivitätstyp A zugeordnet.

- *add-repetitively(A, t)@T*:

  Eine Aktivität vom Typ A muss innerhalb des Gültigkeitsintervalls *T* wiederholt zusätzlich ausgeführt werden. Der zeitliche Abstand zwischen zwei Aktivitäten vom Typ A beträgt *t*, wobei t eine relative Zeitangabe wie z. B. *zwei Minuten* oder *drei Stunden* ist.

  Die Umsetzung dieser Kontroll-Aktion erfolgt nicht durch den Adaptations-Agenten, sondern über die Generierung eines neuen Workflows. Dieser enthält eine Schleife, in der die Aktivität vom Typ A im Abstand *t* solange wiederholt ausgeführt wird, bis das Gültigkeitsintervall *T* verstrichen ist. Diese Kontroll-Aktion wird in den weiteren Kapiteln nicht berücksichtigt.

- *delay(A, t)@T*:

  Die Ausführung einer Aktivität vom Typ A wird um den Zeitraum *t* verzögert. Die Verzögerung wird auch dann durchgeführt, wenn die Aktivität dadurch erst nach dem Ende von *T* ausgeführt werden wird.

Der Adaptations-Agent muss also die Ausführung aller Knoten der Workflow-Instanz mit dem Aktivitätstyp A, die innerhalb des Gültigkeitsintervalls *T* voraussichtlich ausgeführt werden, um *t* verzögern.

- *check(A)@T*:
  Die *check*-Kontroll-Aktion wird immer dann gefolgert, wenn Aktivitäten vom Typ A von einem logischen Fehler betroffen sind, aber keine der anderen Kontroll-Aktionen eindeutig gefolgert werden konnte. Das ist z. B. dann der Fall, wenn das weitere Vorgehen und damit die durchzuführende Kontroll-Aktion vom jeweiligen Patienten abhängt und nicht allgemein bestimmt werden kann.
  Feuert z. B. folgende Regel: *Wenn der Leukozytenwert zwischen 1500 und 2500 liegt, überprüfe die Gabe von Etoposid*, wird eine *check*-Kontroll-Aktion gefolgert. Diese veranlasst den Adaptations-Agenten, dem Arzt alle Knoten mit dem Aktivitätstyp *Verabreiche Etoposid* anzuzeigen, die innerhalb des Gültigkeitsintervalls *T* zur Ausführung kommen werden. Dieser spezifiziert dann, ob der Patient das Medikament Etoposid weiterhin bekommen soll, ob es abgesetzt oder durch ein anderes ersetzt werden muss.

Das Gültigkeitsintervall $T=(T_{start}, T_{end})$ liegt auf einer Zeitachse mit diskreten Werten. Die kleinste Einheit der Achse hängt vom Anwendungsgebiet ab. In der Onkologie können es zum Beispiel Tage oder Stunden sein, während in anderen Gebieten wie der Intensiv-Medizin vielleicht Minuten oder Sekunden als kleinste Einheit nötig sind.
Die Intervalle, die an den Adaptations-Agenten übergeben werden, bestehen nur aus relativen Werten. Falls die Grenzen eines Gültigkeitsintervalls absolute Werte sind (z. B. 05. April), werden diese in Angaben relativ zum Zeitpunkt *now* umgerechnet. Das Symbol *now* bezeichnet den Zeitpunkt, an dem der Fehler entdeckt und der Workflow unterbrochen wurde[1]. Angenommen das aktuelle Datum wäre der 03. April, dann würde das Intervall (*now*, 05.April) in (*now*, 2 days) umgerechnet werden.
Die obere Intervallgrenze T_{end} kann auch unendlich sein. Dies bedeutet, dass eine Kontroll-Aktion für alle betroffenen Aktivitäten in dem noch auszuführenden Teil des Workflows gilt und für alle zukünftig gestarteten Workflows. Ist bei einem Patienten z. B. eine Allergie gegen

1. Es wird angenommen, dass die Zeit zwischen der Entdeckung des Fehlers und der Unterbrechung des Workflows vernachlässigbar ist.

einen bestimmten Wirkstoff aufgetreten, müssen alle Workflows, die zur Behandlung dieses Patienten ausgeführt werden, direkt nach der Initialisierung überprüft werden, ob darin die Verabreichung des betroffenen Wirkstoffs vorgesehen ist.
Als Intervallgrenzen eines Gültigkeitsintervalls können auch Bedingungen stehen, beispielsweise *bis der Wert x>5000*. In diesem Fall ist nicht bekannt, zu welchem zukünftigen Zeitpunkt relativ zu *now* die Bedingung erfüllt sein wird. Eine Abschätzung, welcher Teil des Workflows durchlaufen werden wird, bis die Bedingung erfüllt ist, wie sie bei der prädiktiven Strategie erforderlich wäre, ist deshalb nicht möglich. Kontroll-Aktionen, deren Gültigkeitsintervall durch solche Bedingungen begrenzt ist, können deshalb nur mit der reaktiven Adaptations-Strategie bearbeitet werden, bei der der Adaptations-Agent nur Änderungen an der betroffenen Workflow-Instanz vornimmt.
Die unterschiedlichen Adaptations-Strategien werden im Abschnitt 3.3 genauer erläutert.

3.2 Sortierung der Kontroll-Aktionen

Für eine von einem logischen Fehler betroffene Instanz können eine oder mehrere Kontroll-Aktionen gefolgert worden sein. Wenn diese unterschiedliche Gültigkeitsintervalle haben, muss für jedes Gültigkeitsintervall die Adaptations-Region bestimmt werden, also der Teil der Workflow-Instanz, der voraussichtlich innerhalb dieses Intervalls durchlaufen werden wird.
Überschneiden sich einige der Gültigkeitsintervalle, kann durch eine Sortierung der Kontroll-Aktionen nach ihren Gültigkeitsintervallen erreicht werden, dass die Adaptations-Region nicht für jedes Gültigkeitsintervall neu berechnet werden muss, sondern jeweils nur einmal für eine Gruppe von Kontroll-Aktionen. Wie diese Gruppen gebildet werden, wird im Folgenden dargestellt.
Wurden eine Kontroll-Aktion oder mehrere Kontroll-Aktionen mit dem gleichen Gültigkeitsintervall T gefolgert, gibt es nur eine Gruppe. Die Adaptations-Region muss einmal für dieses Gültigkeitsintervall T bestimmt werden.
Gelten mehrere Kontroll-Aktionen, von denen mindestens zwei unterschiedliche Gültigkeitsintervalle haben, die sich überschneiden, können die Kontroll-Aktionen in Gruppen eingeordnet werden. Dabei werden die Gültigkeitsintervalle so in Teilintervalle unterteilt, dass immer dort

ein neues Teilintervall beginnt, wo ein Überschneidungspunkt zwischen zwei Gültigkeitsintervallen liegt.

Zur Verdeutlichung soll folgendes Beispiel dienen (siehe Abbildung 3-1): Angenommen, es gibt drei Kontroll-Aktionen mit unterschiedlichen Gültigkeitsintervallen, die alle bei *now* beginnen und bis zu den oberen Grenzen T_1, T_2 und T_3 mit $T_1 < T_2 < T_3$ reichen. Dann bildet (now, T_1) das erste Teilintervall, (T_1, T_2) das zweite Teilintervall und (T_2, T_3) das dritte Intervall.

Im ersten Intervall sind alle Kontroll-Aktionen enthalten, im zweiten nur noch die zweite und die dritte und im letzten Intervall ist nur die dritte Kontroll-Aktion enthalten.

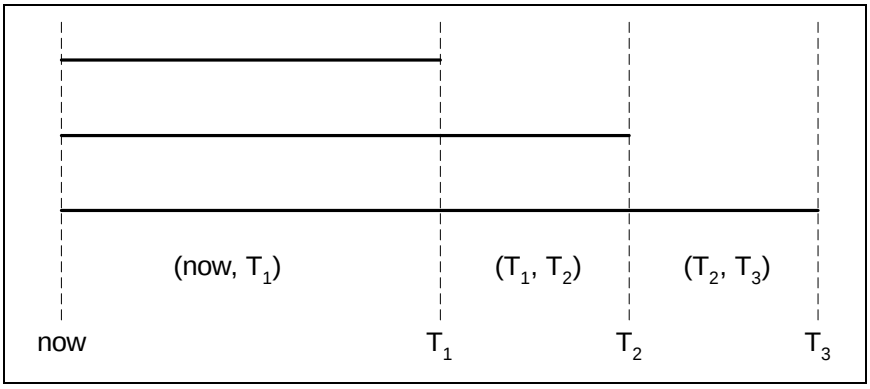


Abb. 3-1: Aufteilung mehrerer Gültigkeitsintervalle in Teilintervalle.

Ist unter den Kontroll-Aktionen eine *add*-Kontroll-Aktion, ist folgendes zu beachten: Da der neue Knoten nur einmal zusätzlich eingefügt werden darf, darf eine solche Kontroll-Aktion nur solange mit berücksichtigt werden, bis der Knoten erfolgreich eingefügt werden konnte. Für spätere Intervalle darf sie nicht mehr bearbeitet werden.

Angenommen im obigen Beispiel wäre eine *add*-Kontroll-Aktion mit dem Gültigkeitsintervall (now, T_3) gefolgert worden. Dann wird diese Kontroll-Aktion zuerst im ersten Intervall (now, T_1) bearbeitet. Wenn der Knoten hier bereits erfolgreich eingefügt werden kann, darf diese Kontroll-Aktion in den beiden restlichen Intervallen nicht mehr bearbeitet werden, obwohl sich ihr Gültigkeitsintervall auch mit den Intervallen (T_1, T_2) und (T_2, T_3) überschneidet.

Somit kann im Folgenden angenommen werden, dass mehrere Kontroll-Aktionen für das gleiche Gültigkeitsintervall gelten. Zunächst wird nun angenommen, dass pro Intervall nur eine Kontroll-Aktion gilt. Unter dieser Beschränkung werden die möglichen Strategien vorgestellt, nach denen die Adaptationen vorgenommen werden können. Der Fall, dass mehrere Kontroll-Aktionen für ein Intervall gelten, wird im Kapitel 5 behandelt.

3.3 Adaptations-Strategien

Für die Adaptation eines Workflows zur Laufzeit gibt es zwei mögliche Strategien: eine reaktive und eine prädiktive Strategie. Im Folgenden werden beide vorgestellt und ihre Vor- und Nachteile aufgezeigt. Anschließend wird erläutert, mit welcher Strategie das Workflow-System AGENTWORK arbeitet.

- *Reaktive Strategie:*
  Bei der reaktiven Strategie wird mit der Adaptation gewartet, bis der Kontrollfluss eine von einem logischen Fehler betroffene Aktivität erreicht, d. h. der betroffene Knoten in den Zustand Control-Activated (vgl. Abschnitt 2.3.2) übergeht. Dann unterbricht die Workflow-Engine die Instanz und informiert den Kontroll-Agenten darüber. Dieser teilt dem Adaptations-Agenten die ID des betroffenen Knotens und die Kontroll-Aktion mit. Der Adaptations-Agent führt die Änderung durch, lässt sie vom Nutzer bestätigen und informiert dann die Engine, dass der Workflow weiter ausgeführt werden kann.
  Der Workflow läuft dann solange weiter bis wieder eine von einem logischen Fehler betroffene Aktivität vor der Ausführung steht. Dann wird wieder unterbrochen, die Änderung durchgeführt und dann fortgesetzt.
  Kann die Kontroll-Aktion auch ohne Änderungen am Kontroll- bzw. Datenfluss direkt durch die Engine bearbeitet werden, ist keine Unterbrechung erforderlich. Beispielsweise kann bei einer *drop*-Kontroll-Aktion der betroffene Knoten einfach übersprungen werden, wenn der Knoten keine ein- oder ausgehende Datenflusskanten hat. Der Kontrollfluss geht zu dessen Nachfolger über.

Ein Nachteil der reaktiven Strategie ist, dass die Instanz jedesmal erneut unterbrochen werden muss, wenn der Kontrollfluss einen Knoten erreicht, dem der betroffene Aktivitätstyp zugeordnet ist. Wenn das Gültigkeitsintervall *T* der Kontroll-Aktion sehr lang ist und der betroffene Aktivitätstyp sehr häufig darin vorkommt, muss die Instanz oft unterbrochen werden.

Ein weiterer Nachteil ist, dass die Änderungen erst dann vorgenommen werden, wenn die betroffene Aktivität ausgeführt werden soll. Das kann z. B. problematisch werden, wenn für eine durch eine *replace*-Kontroll-Aktion neu eingefügte Aktivität vorbereitende Aktivitäten notwendig sind, die nicht elektronisch im Workflow-System erfasst sind, oder wenn eine Aktivität gelöscht wird (durch eine *drop*-Kontroll-Aktion), für die schon vorbereitende Aktivitäten durchgeführt wurden, die nicht im Workflow erfasst sind. Die Wahrscheinlichkeit, dass solche nicht erfassten Aktivitäten nötig sind, ist nicht zu vernachlässigen, da ein Anwendungsgebiet nie vollständig elektronisch erfasst werden kann. Das gilt vor allem in der Anfangsphase des Einsatzes eines Workflow-Systems.

Sind für eine neu eingefügte Aktivität zusätzliche Vorbereitungen nötig, die noch nicht durchgeführt wurden, wenn die Aktivität zur Ausführung ansteht, verzögert sich die weitere Ausführung des Workflows. Wenn zum Beispiel ein anderes Medikament verabreicht werden soll, dieses aber erst bestellt werden muss, kann die Aktivität erst ausgeführt werden, wenn es da ist.

Wurde eine Aktivität (z. B. die Verabreichung eines bestimmten Medikaments) erst direkt vor ihrer Ausführung gelöscht, wurde das Medikament eventuell schon bestellt und geliefert, obwohl es gar nicht mehr gegeben werden darf. Wenn das Mittel speziell für einen bestimmten Patienten gemischt wurde, kann es auch nicht anderweitig verwendet werden und muss vernichtet werden. Da Spezialmedikamente nicht billig sind, ist die Vermeidung der unnötigen Bestellung auch eine Kostenfrage.

- *Prädiktive Strategie:*

  Bei der prädiktiven Strategie wird die betroffene Workflow-Instanz sofort unterbrochen, wenn ein Ereignis einen logischen Fehler verursacht hat und eine Kontroll-Aktion gefolgert wurde, deren Gültigkeitsintervall *T* durch einen expliziten Zeitwert begrenzt ist.

  Der Adaptations-Agent schätzt dann zunächst ab, welcher Teil der noch nicht ausgeführten Workflow-Definition wahrscheinlich bis zum Ende des Gültigkeitsintervalls *T* durch-

laufen werden wird. Danach überprüft er, ob darin Knoten vorkommen, denen der betroffene Aktivitätstyp zugeordnet ist, und wenn ja, führt er für diese die notwendigen Adaptationen durch. Abschließend läßt er die Änderungen vom Nutzer bestätigen und gibt dann die Instanz zur weiteren Ausführung frei.

Im Idealfall muss bei der prädiktiven Strategie die Instanz zur Behandlung eines logischen Fehlers nur einmal unterbrochen werden, da bereits dann alle Änderungen durchgeführt wurden. Eine erneute Unterbrechung ist nur nötig, wenn die Bestimmung der Knoten, die voraussichtlich im Gültigkeitsintervall *T* ausgeführt werden, nicht korrekt war. In diesem Fall müssen Änderungen nachgeholt oder rückgängig gemacht werden.
Ein weiterer Vorteil der prädiktiven Strategie ist, dass alle Änderungen an betroffenen Knoten am Anfang des Gültigkeitsintervalls durchgeführt und vor der weiteren Ausführung des Workflows dem Benutzer angezeigt werden. So können vorbereitende Aktivitäten zu neu eingefügten Aktivitäten, die nicht im Workflow-System erfasst sind, rechtzeitig eingeleitet werden. Genauso können vorbereitende Schritte für gelöschte Aktivitäten abgebrochen oder gar nicht erst begonnen werden. Der Anwender wird also von solchen Änderungen nicht überrascht und kann rechtzeitig reagieren. Die weitere Ausführung des Workflows wird nicht unnötig verzögert.

Ein Nachteil der prädiktiven Strategie ist, dass die Abschätzung der Adaptations-Region beispielsweise durch ein sehr langes Gültigkeitsintervall sehr ungenau werden kann. Durch die ungenaue Abschätzung wächst die Wahrscheinlichkeit, dass zu viele oder zu wenige Änderungen vorgenommen werden. Damit wächst auch die Wahrscheinlichkeit für eine erneute Unterbrechung der Instanz, um die fehlerhaften Adaptationen zu korrigieren oder weitere Adaptationen nachzuholen. In diesem Fall sollte zur reaktiven Strategie übergegangen werden.
Diese kann auch dann eingesetzt werden, wenn keine Abschätzung des betroffenen Teils des Workflows möglich ist, weil das Ende des Gültigkeitsintervalls *T* nur implizit durch eine Bedingung (z. B. $x < 5000$) gegeben ist.

Das Workflow-System AGENTWORK wendet eine Kombination aus beiden Strategien an, um die Vorteile von beiden nutzen zu können und die Nachteile möglichst zu vermeiden.

Wenn die Grenzen des Gültigkeitsintervalls durch Bedingungen gegeben sind, wird die reaktive Strategie verwendet. Ansonsten wird zuerst immer mit der prädiktiven Strategie gearbeitet. Diese wird nur dann aufgegeben, wenn die Abschätzung zu ungenau wird und damit die Wahrscheinlichkeit steigt, den Workflow erneut unterbrechen zu müssen, um Änderungen zurückzunehmen oder nachzuholen. In diesem Fall ändert der Adaptations-Agent nur den Teil, für den die Abschätzung hinreichend gut ist, und informiert den Kontroll-Agenten, dass für den Rest des Gültigkeitsintervalls mit der reaktiven Strategie zu arbeiten ist.

4. Bestimmung der Adaptations-Region

Die erste Teilaufgabe des Adaptations-Agenten ist die Bestimmung der *Adaptations-Region*. Diese wird folgendermaßen definiert:

> Die *Adaptations-Region* AR_T ist der Teil einer Workflow-Instanz, der innerhalb eines Zeitintervalls *T* voraussichtlich ausgeführt werden wird.
>
> Ein Knoten gehört dann zur Adaptations-Region, wenn der Beginn seiner Ausführung innerhalb des Intervalls *T* liegt.

Die Berechnung erfolgt unter Verwendung der *Ausführungsdauern* der Aktivitätsknoten und der *Wartezeiten* der Kommunikationsknoten und temporalen Transitionen.
Als Ausführungsdauer bezeichnet man die Zeit, die nötig ist, um eine einzelne Aktivität oder eine Folge von Aktivitäten auszuführen. Die Wartezeit eines Kommunikationsknotens ist die Zeit, die vergeht, bis ein Kommunikationsknoten seine Daten empfangen oder verschickt hat.
Voraussichtlich bedeutet in der Definition, dass die Adaptations-Region dann vollständig innerhalb *T* durchlaufen wird, wenn die Werte für die Ausführungsdauern und Wartezeiten möglichst nah an den realen Werten liegen und keine technischen Fehler auftreten, die die Ausführung der Instanz verzögern. Solche Fehler wären zum Beispiel ein Rechnerausfall oder der Ausfall einer Datenbank.
In einer Workflow-Definition werden die auszuführenden Aktivitäten durch Aktivitätsknoten repräsentiert. Während ihrer Ausführung durchlaufen diese Knoten die Zustände Control-Activated, Object-Activated und Currently-Processed (zu Knotenzuständen vgl. Abschnitt 2.3.2). Jeder der drei Zustände hat eine signifikante Dauer, die bei der Bestimmung der Adaptations-Region berücksichtigt werden muss. Deshalb umfasst die Ausführungsdauer einer Aktivität (bzw. eines Aktivitätsknotens) die Zeitspanne von der Aktivierung des Knotens durch den Kontrollfluss bis zum Abschluss der Ausführung der ihm zugeordneten Aktivität. In dieser Zeit ist auch die Zeit zur Beschaffung eventuell benötigter Eingabedaten und zur Speicherung von Ausgabedaten eingeschlossen.

Für Ausführungsdauern und Wartezeiten können minimale, durchschnittliche oder maximale Werte verwendet werden. Diese drei Alternativen werden zunächst genauer erläutert und ihre Auswirkungen auf die Adaptations-Region AR_T werden diskutiert. Danach werden die Algorithmen zur Berechnung von AR_T detailliert vorgestellt. Anschließend wird erläutert, wie das Ende der Adaptations-Region AR_T überwacht wird, und wie man die Exaktheit der Abschätzung beeinflussen kann. Zum Schluss wird dargestellt, wie auf technische Fehler (z. B. Datenbankausfälle) während der Bestimmung von AR_T reagiert wird.

4.1 Zeittypen und ihre Auswirkung auf die Adaptations-Region

Je nachdem ob man für Ausführungsdauern und Wartezeiten minimale, durchschnittliche oder maximale Werte verwendet, erhält man unterschiedliche Adaptations-Regionen AR_T. Auch die Auswirkungen auf die weitere Ausführung des adaptierten Workflows sind unterschiedlich. Im folgenden Abschnitt werden die verschiedenen Möglichkeiten und ihre Auswirkungen auf die Adaptations-Region AR_T vorgestellt.

4.1.1 Zeittypen

Man kann für Ausführungsdauern und Wartezeiten minimale, maximale oder durchschnittliche Werte verwenden. Die Tabelle 4-1 gibt eine Überblick über die verschiedenen Möglichkeiten und über die Herkunft der Werte.

Zeittyp	Untertyp	Erstellungsart
minimale Dauer	absolutes Minimum (kleinste anzunehmende Minimalzeit)	Angabe zur Buildtime durch Workflow-Modellierer
	Mittelung der minimalen Werte	Berechnung zur Runtime aus Mitschnitten der Engine
maximale Dauer	absolutes Maximum (größte anzunehmende Maximalzeit)	Angabe zur Buildtime durch Workflow-Modellierer
	Mittelung der maximalen Werte	Berechnung zur Runtime aus Mitschnitten der Engine
durchschnittliche Dauer		Angabe zur Buildtime durch Workflow-Modellierer **und** Berechnung zur Runtime aus Mitschnitten der Engine

Tabelle 4-1: Zeittypen und ihre Herkunft.

Die minimale Dauer gibt an, wie lange die Ausführung einer Aktivität im kürzesten Fall dauert bzw. wie lange ein Kommunikationsknoten im kürzesten Fall warten muss. Der zweite Typ, die maximale Dauer, gibt die Ausführungsdauer bzw. die Wartezeit für den längsten Fall an. Bei beiden Typen gibt es zwei Untertypen: das absolute Minimum/Maximum und die Mittelung aller bisher aufgetretenen Minimal-/Maximalwerte.

Der absolute Wert wird während der Modellierung des Workflows angegeben und stellt die kürzeste bzw. längste anzunehmende Ausführungsdauer bzw. Wartezeit dar. Wenn dieser Wert nur sehr selten auftritt und die Ausführung meistens länger bzw. kürzer dauert, wird für die Abschätzung meistens ein zu kleiner bzw. zu großer Wert verwendet. Die Abschätzung wird also ungenauer. Um das zu verhindern, kann die Engine während der Ausführung eines Workflows die reellen Werte für minimale/maximale Ausführungsdauern und Wartezeiten mitschneiden und aus diesen gemittelte Minimal-/Maximalwerte berechnen. Diese bilden den tatsächlichen Wert genauer ab und machen so auch die Abschätzung genauer.

Die durchschnittliche Dauer kommt in den meisten Fällen der Realität am nächsten. Die Werte werden zuerst bei der Modellierung des Workflows als Schätzwerte (z. B. aus Statistiken)

angegeben. Während der Ausführung von Workflows werden sie (wie gemittelte Minimal- oder Maximalwerte) durch von der Engine mitgeschnittene Werte der Realität weiter angenähert.

In den meisten Anwendungsgebieten ist es wahrscheinlich sinnvoll, mit den durchschnittlichen Dauern zu arbeiten, da sie die Realität am besten abbilden. Da es aber in manchen Anwendungsgebieten durchaus erforderlich sein kann, die - eventuell gemittelten - Minimal- oder Maximalwerte zu verwenden, sollte die Entscheidung dem Anwender überlassen werden. Er kann dann zum Beispiel bei der Installation angeben, ob er für die Ausführungsdauern und Wartezeiten den minimalen, durchschnittlichen oder maximalen Wert verwenden möchte.

4.1.2 Typen von Adaptations-Regionen

Es gibt drei unterschiedliche Typen von Adaptations-Regionen AR_T, abhängig davon, ob man minimale, maximale oder durchschnittliche Werte für die Ausführungsdauern und die Wartezeiten verwendet.

- *Maximale Adaptations-Region*
  Mit den absolut minimalen Dauern erhält man die maximale Adaptations-Region AR_T. Das ist der Teilgraph, der in der Zeit *T* maximal durchlaufen werden kann, wenn jede Aktivität schnellstmöglich ausgeführt wird, jeder Kommunikationsknoten nur minimal warten muss und bei temporalen Transitionen (d. h. mit der Semantik *warte mindestens x aber höchstens y Zeiteinheiten*) immer die untere Intervallgrenze herangezogen wird.
  Verwendet man gemittelte Minimalwerte erhält man nicht mehr die maximale Adaptations-Region, sondern eine, die etwas kleiner ist.

- *Minimale Adaptations-Region*
  Die minimale Adaptations-Region AR_T erhält man bei Verwendung der absolut maximalen Dauern. Sie entspricht dem Teilgraph der Workflow-Definition, der im Intervall *T* abgesehen von technischen Fehlern (z. B. einem Datenbankausfall) auf jeden Fall durchlaufen werden wird, da hier für jede Aktivität, jeden Kommunikationsknoten und jede

temporale Transition die maximale Dauer angenommen wird. Verwendet man gemittelte Maximalwerte, erhält man nicht mehr die minimale Adaptations-Region, sondern eine, die etwas größer ist.

- *Durchschnittliche Adaptations-Region*
  Wenn man für Ausführungsdauern und Wartezeiten die durchschnittlichen Werte verwendet, erhält man die durchschnittliche Adaptations-Region AR_T. Diese entspricht dem Teilgraph der Workflow-Definition, der innerhalb *T* im Durchschnitt durchlaufen werden wird.

4.1.3 Auswirkungen des Zeittyps auf die Adaptations-Region

Abhängig von der Wahl des Zeittyps für die Ausführungsdauern und Wartezeiten ergeben sich unterschiedliche Auswirkungen auf die Adaptations-Region AR_T. Jeder Typ hat Vor- und Nachteile, die im Folgenden diskutiert werden.

- *Absolut minimale Ausführungsdauer*
  Mit den absolut minimalen Ausführungsdauern und Wartezeiten wird die maximale Adaptations-Region AR_T bestimmt. Wenn die tatsächliche Ausführung einer oder mehrerer Aktivitäten länger dauert als die minimale Zeit angibt, kommen unter Umständen nicht mehr alle Knoten, die in AR_T liegen, innerhalb des Intervalls *T* zur Ausführung.
  In Abbildung 4-1 ist dieser Fall dargestellt. Die maximale Adaptations-Region AR_T umfasst die Knoten 1 bis 10. Wenn von den Knoten 2 bis 6 oder 8 mindestens einer länger braucht als die minimale Zeit, kann es sein, dass Knoten 9 und 10 oder nur Knoten 10 nicht mehr innerhalb des Gültigkeitsintervalls *T* ausgeführt werden.
  Wenn einer dieser beiden Knoten von der Kontroll-Aktion betroffen war, für die AR_T bestimmt wurde, wurde er geändert. Deshalb muss diese Änderung rückgängig gemacht werden, da die tatsächliche Ausführung des Knotens nicht mehr innerhalb des Gültigkeitsintervalls *T* erfolgt.

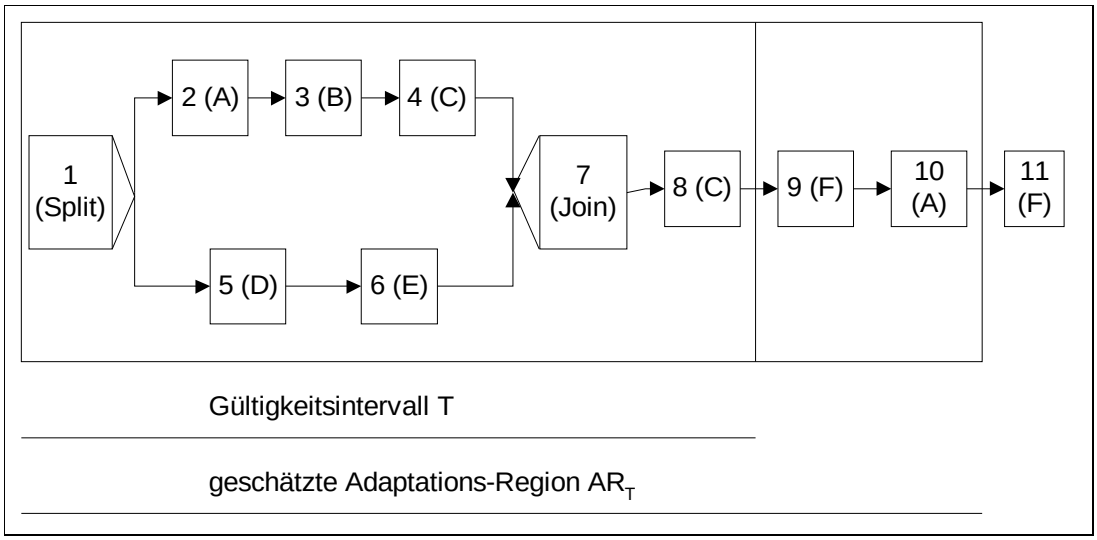


Abb. 4-1: Ausführung des Workflows langsamer als berechnet.

Im Allgemeinen ist bei einer falschen Abschätzung von AR_T nur eine erneute Unterbrechung nötig, da man einfach warten kann, bis das Gültigkeitsintervall verstrichen ist, und dann für alle noch nicht ausgeführten aber geänderten Knoten die Änderungen rückgängig macht. Dabei ist keine erneute Bestimmung von AR_T nötig.

Bei diesem Vorgehen ist bei einer *add*-Kontroll-Aktion folgendes Problem zu berücksichtigen:

Eine Aktivität soll innerhalb *T* zusätzlich ausgeführt werden, die einzige passende Stelle befindet sich am Ende der geschätzten Adaptations-Region AR_T. Wenn der Workflow langsamer läuft als erwartet, wird die neue Aktivität nicht rechtzeitig ausgeführt. Eine Lösung besteht darin, mit dem Rückgängigmachen der Änderungen nicht bis zum Ende von *T* sondern bis zu einem Zeitpunkt $T' < T$ zu warten, der so gewählt ist, dass die hinzuzufügende Aktivität im Intervall (T', T) noch ausgeführt werden kann. Allerdings müsste dann am Ende von *T* die Workflow-Instanz noch einmal unterbrochen werden, um die übrigen Änderungen (basierend auf anderen Kontroll-Aktionen) rückgängig zu machen.

- *Absolut maximale Ausführungsdauer*

  Werden die absolut maximalen Ausführungsdauern und Wartezeiten verwendet, erhält man die minimale Adaptations-Region AR_T. Wenn mindestens eine Aktivität schneller

ausgeführt wird als erwartet, können weitere Knoten innerhalb des Gültigkeitsintervalls ausgeführt werden, die aber in AR_T nicht enthalten sind.

In Abbildung 4-2 werden die ursprünglich zu AR_T gehörenden Knoten 1 bis 8 in kürzerer Zeit durchlaufen als bei der Bestimmung von AR_T angenommen, dadurch kommen auch die Knoten 9 und 10 noch während des Gültigkeitsintervalls *T* zur Ausführung.

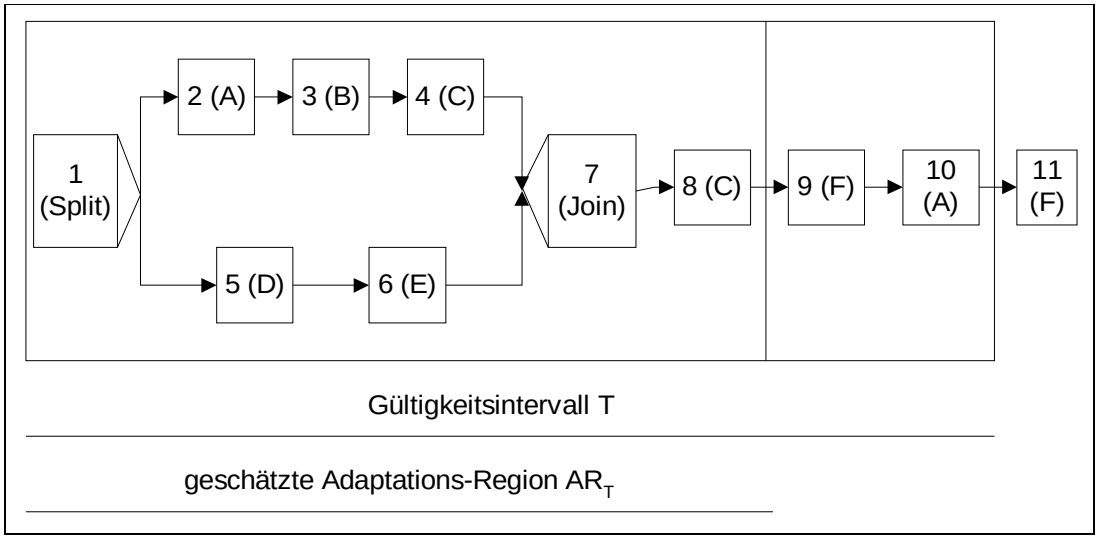


Abb. 4-2: Ausführung des Workflows schneller als berechnet.

Ist das Ende von AR_T zum Zeitpunkt $T' < T$ erreicht, müssen für das Intervall (T', T) erneut AR_T bestimmt und Änderungen durchgeführt werden. Dabei kann es sein, dass man das Intervall wieder nicht voll ausschöpft und erneut nachadaptieren muss. Es kann also zu mehreren nachfolgenden Unterbrechungen der Instanz kommen.

Wie in Abschnitt 4.1.1 beschrieben, können die absolut minimalen oder absolut maximalen Dauern durch gemittelte Minimal- oder Maximalwerte ersetzt werden. Damit ist aber nicht mehr garantiert, dass man bei Verwendung der minimalen Ausführungsdauern immer nur Änderungen rückgängig machen muss und bei Verwendung der maximalen Ausführungsdauern immer nur erneut abschätzen muss.

- *Durchschnittliche Ausführungsdauer*

  Wenn man die durchschnittlichen Ausführungsdauern und Wartezeiten verwendet, erhält man den Teil der Instanz, der im Durchschnitt innerhalb *T* ausgeführt wird. Der Workflow kann also sowohl langsamer als auch schneller laufen als erwartet. Man muss im Allgemeinen genauso oft Änderungen rückgängig machen wie erneut abschätzen und Änderungen nachholen.

  Zusätzlich ist wie bei den minimalen Ausführungsdauern der Fall zu berücksichtigen, dass Aktivitäten, die durch eine *add*-Kontroll-Aktion neu eingefügt wurden, eventuell nicht rechtzeitig ausgeführt werden. Man kann diesem Problem aber wiederum dadurch begegnen, dass man zu einem Zeitpunkt $T' < T$ überprüft, ob alle neu einzufügenden Aktivitäten ausgeführt wurden und wenn nicht, diese im Intervall (T', T) ausführt. T' ist dabei so zu wählen, dass alle neu einzufügenden Aktivitäten innerhalb dieses Intervalls ausgeführt werden können.

Zusammenfassend kann man feststellen, dass keine der drei Alternativen signifikant mehr Vorteile hat als die anderen. Verwendet man absolut oder gemittelt maximale oder durchschnittliche Werte, muss die Instanz möglicherweise mehrfach unterbrochen werden. Außerdem entsteht zusätzlicher Rechenaufwand durch die erneute Abschätzung der Adaptations-Region AR_T.

Bei absolut minimalen oder maximalen Werten besteht die Gefahr, nie eine passende Adaptations-Region zu bekommen, da der Wert eine einmalige Ausnahme gewesen sein kann. Versucht man diesen Effekt durch Mittelung zu mildern, kann es auch bei minimalen Werten zu mehreren Unterbrechungen und dem Aufwand erneuter Abschätzungen kommen.

Der Adaptations-Agent arbeitet deshalb mit den durchschnittlichen Ausführungsdauern und Wartezeiten von Knoten, da diese die Realität besser abbilden als möglicherweise nur einmal aufgetretene Minimal- oder Maximalwerte. Für temporale Transitionen mit einer Intervallangabe wird das arithmetische Mittel der beiden Grenzwerte genommen.

Im Folgenden sind immer die durchschnittliche Adaptations-Region AR_T und die durchschnittliche Ausführungsdauer bzw. Wartezeit gemeint, wenn der Begriff Adaptations-Region, Ausführungsdauer bzw. Wartezeit verwendet wird.

4.2 Berechnung der Adaptations-Region

In den folgenden Abschnitten werden die Algorithmen zur Berechnung der gesamten Adaptations-Region AR_T sowie zur Bestimmung der Ausführungsdauer und Knotenmenge der einzelnen Kontroll-Regionen vorgestellt.

4.2.1 Bestimmung der gesamten Adaptations-Region

Zuerst wird der Algorithmus für die Bestimmung der gesamten Adaptations-Region AR_T erläutert. Ein vereinfachtes Flussdiagramm dazu ist in Abbildung 4-3 zu sehen. Die ausführlichen Diagramme befinden sich im Anhang B.

Beim Aufruf des Agenten wird diesem die *Unterbrechungsmenge U* übergeben. Diese enthält die eindeutigen Bezeichnungen der Knoten (im Folgenden auch IDs genannt), an denen die von einem logischen Fehler betroffene Workflow-Instanz unterbrochen wurde. Die Menge enthält immer mindestens einen Knoten. Wurden zum Zeitpunkt der Unterbrechung mehrere Pfade parallel ausgeführt, kann sie auch mehrere Knoten enthalten.
Die Knoten der Unterbrechungsmenge bestimmen die Pfade, die bei der weiteren Ausführung des Workflows durchlaufen werden, und damit auch den Beginn der Adaptations-Region AR_T.

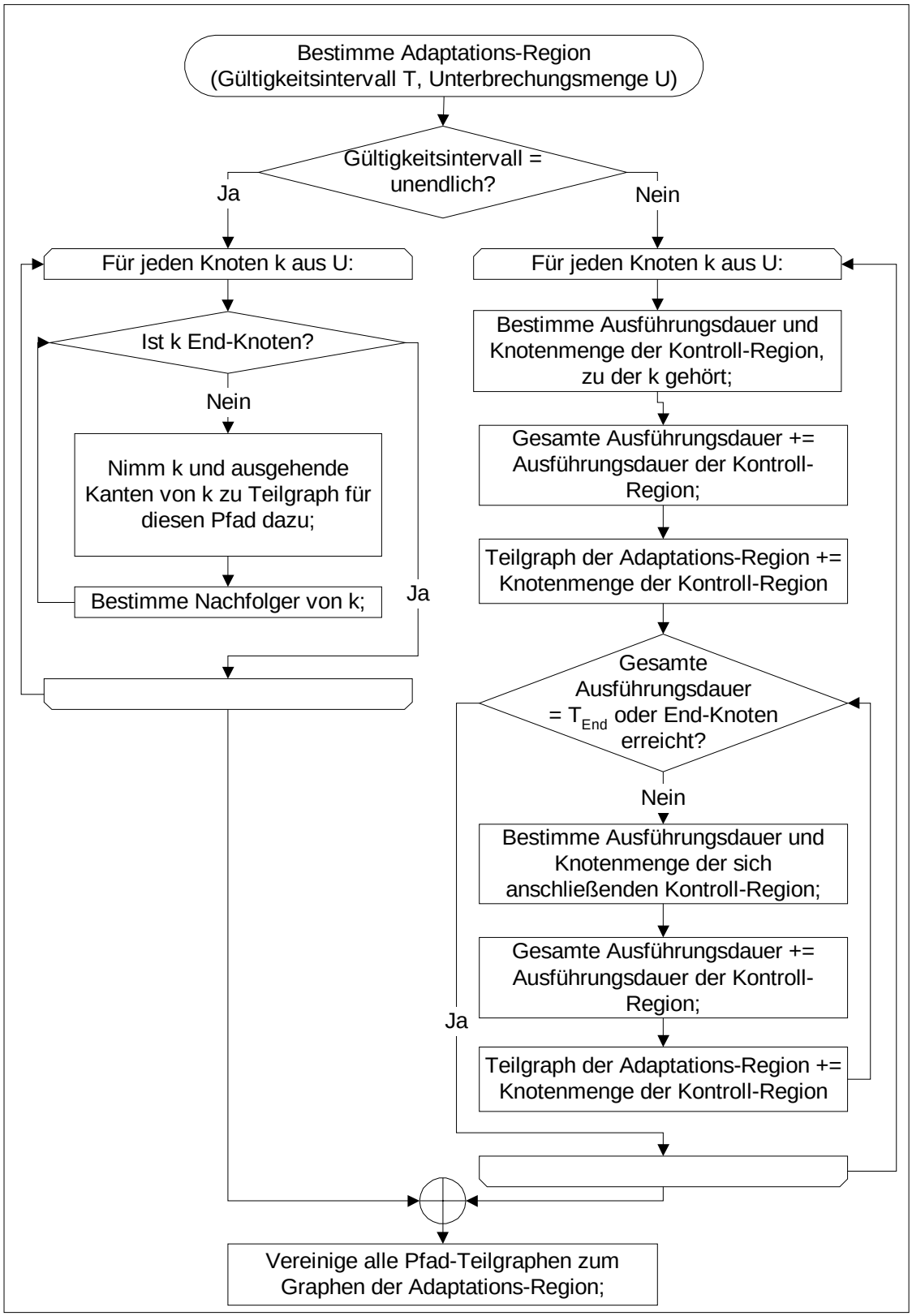


Abb. 4-3: Algorithmus zur Bestimmung der gesamten Adaptations-Region AR_T.

Der Algorithmus hat zwei alternative Pfade. Wenn das Gültigkeitsintervall *T* unendlich ist, gilt die Kontroll-Aktion für alle noch nicht ausgeführten Teile der Instanz. In diesem Fall muss der Adaptations-Agent nur den Restgraphen der Workflow-Instanz bestimmen, der noch nicht ausgeführt wurde. Dabei müssen keine Ausführungsdauern oder Wartezeiten berücksichtigt werden. Ausgehend von den Knoten der Unterbrechungsmenge bestimmt der Agent jeden möglichen Pfad bis zum Endknoten der Workflow-Instanz. Die Teilgraphen, die diese Pfade repräsentieren, werden anschließend zum Graphen der Adaptations-Region AR_T vereinigt.

Wenn die obere Grenze des Gültigkeitsintervalls *T* einen Wert T_{End} kleiner unendlich hat, müssen bei der Bestimmung von AR_T die Ausführungsdauern der Aktivitäten und die Wartezeiten der Kommunikationsknoten und temporalen Transitionen berücksichtigt werden, da AR_T möglicherweise nur einen Teil der noch nicht ausgeführten Workflow-Instanz umfasst. Wenn das Gültigkeitsintervall lang genug ist, kann AR_T auch bis zum Ende-Knoten reichen. Da das aber zum Zeitpunkt der Bestimmung von AR_T nicht bekannt ist, wird im Fall eines endlichen Gültigkeitsintervalls wie folgt vorgegangen:
Die Knoten der Unterbrechungsmenge U bestimmen die Pfade, die bei der weiteren Ausführung der Instanz durchlaufen werden. Jeder dieser Pfade besteht aus einer oder mehreren Kontroll-Regionen wie sie in Abschnitt 2.3.1 beschrieben wurden (z. B. aus einer Kombination Sequenz - Split/Join-Region - Sequenz). Jede dieser Kontroll-Regionen hat eine bestimmte Ausführungsdauer und umfasst eine bestimmte Menge von Knoten.
Um die Ausführungsdauer und die Knoten eines Pfades zu bestimmen, werden rekursiv die Ausführungsdauern und Knotenmengen der einzelnen Kontroll-Regionen berechnet. Die Ausführungsdauern der Kontroll-Regionen werden dabei in der Reihenfolge ihres Auftretens addiert. Hat diese Summe die obere Grenze T_{End} des Gültigkeitsintervalls erreicht oder ist der Pfad beim Endknoten des Workflows angekommen, ist auf diesem Pfad das Ende von AR_T erreicht. Die Knoten zwischen dem Knoten aus der Unterbrechungsmenge U, an dem der Pfad beginnt, und dem Knoten, bei dem das Ende von AR_T erreicht werden wird, bilden einen Teilgraphen von AR_T.
Um den gesamten Graph von AR_T zu erhalten, werden die Teilgraphen der einzelnen Pfade vereinigt. Diese Vereinigung bildet einen Teilgraphen der Workflow-Instanz und umfasst alle

Knoten, die entsprechend der Definition der Adaptations-Region AR_T voraussichtlich im Gültigkeitsintervall *T* ausgeführt werden.

In Abbildung 4-4 ist ein Workflow-Ausschnitt dargestellt, in dem gerade zwei Pfade parallel ausgeführt wurden, als die Instanz unterbrochen wurde. Die Unterbrechungsmenge U umfasst also die beiden Knoten 2 und 9. Der Teilgraph von AR_T im oberen Pfad besteht aus den Knoten 2 bis 5, der im unteren Pfad aus den Knoten 9 bis 12. AR_T umfasst also insgesamt die Knoten 2 bis 5 und 9 bis 12.

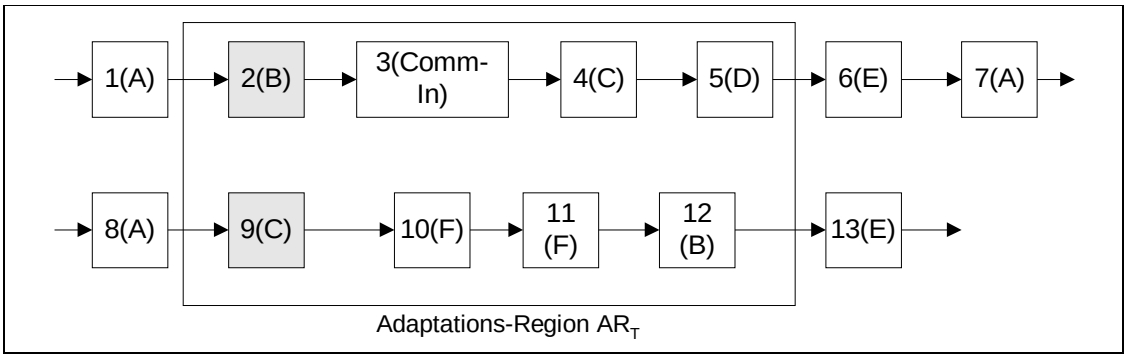


Abb. 4-4: Adaptations-Region AR_T (Knoten 2 und 9 liegen in U).

4.2.2 Berechnung von Ausführungsdauer und Knotenmenge eines Pfades

Der Algorithmus zur Bestimmung der Ausführungsdauer und Knotenmenge eines Pfades arbeitet rekursiv. Wird der Beginn einer neuen Kontroll-Region erreicht, beginnt eine neue Rekursionsstufe, in der die Ausführungsdauer und Knotenmenge der neuen Kontroll-Region bestimmt werden.

Im Folgenden wird dargestellt, wie für die unterschiedlichen Kontroll-Regionen die Ausführungsdauer und Knotenmenge für den Fall eines endlichen Gültigkeitsintervalls bestimmt und wie Synchronisations- und Objektflusskanten dabei berücksichtigt werden.

4.2.2.1 Sequenz

Die einfachste Form einer Kontroll-Region ist die Sequenz. Die einzelnen Knoten werden hier der Reihe nach abgearbeitet, d. h. der nächste Knoten wird aktiviert und ausgeführt, wenn die Ausführung des davor liegenden zu Ende ist. Als Knotentypen können nur Aktivitäts- und Kommunikationsknoten aber keine Kontrollknoten enthalten sein, da diese eine neue Kontroll-Region einleiten. Wenn also in einer Sequenz ein Kontrollknoten (z. B. ein Split-Knoten) auftritt, endet die Sequenz vor diesem Knoten und es beginnt eine neue Kontroll-Region.
In Abbildung 4-5 ist eine einfache Sequenz aus fünf Knoten dargestellt. Die Zahlen unter den Knoten geben die Ausführungsdauern (bei Aktivitätsknoten) bzw. Wartezeiten (bei Kommunikationsknoten) an. Zahlen an Transitionen geben die Wartezeit zwischen den Knoten an, die die Transition verbindet.

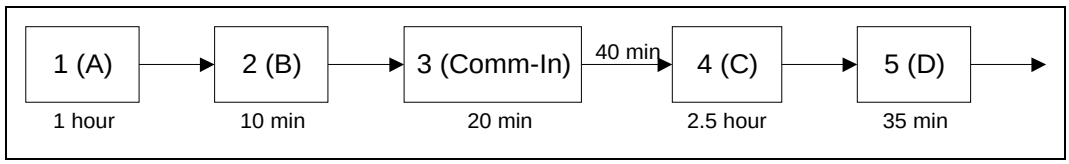


Abb. 4-5: Sequenz.

Die Ausführungsdauer der gesamten Sequenz ist die Summe der Ausführungsdauern und Wartezeiten der in der Sequenz enthaltenen Knoten und Transitionen. Beim Durchlaufen der Sequenz werden diese ausgehend vom ersten Knoten addiert.
Enthält die Sequenz eine Komplex-Aktivität, muss zuerst in einer neuen Rekursionsstufe die Ausführungsdauer des von ihr repräsentierten Subworkflows berechnet werden. Diese wird dann im übergeordneten Workflow zur gesamten Ausführungsdauer der Sequenz addiert.

Eine Sequenz wird solange abgelaufen, bis entweder

- ein Kontrollknoten erreicht wird,
- die Ausführungsdauer der Sequenz T_{End} erreicht hat oder
- der Endknoten des Workflows erreicht wurde.

Im ersten und letzten Fall ist die gesamte Sequenz Teil der Adaptations-Region AR_T, im zweiten Fall nur der Teil, der bis zum Ende von *T* voraussichtlich durchlaufen werden wird.
Wenn T_{End} in einem Subworkflow erreicht wird, gehören der Knoten, der diesen im übergeordneten Workflow repräsentiert, und die Knoten des Subworkflows, die innerhalb des Gültigkeitsintervalls voraussichtlich ausgeführt werden, zu AR_T.

4.2.2.2 Split/Join-Region

Split/Join-Regionen dienen zur Modellierung von Verzweigungen. Sie werden durch einen Split-Knoten am Anfang und einen Join-Knoten am Ende begrenzt. Die unterschiedlichen Arten von Split/Join-Regionen wurden in Abschnitt 2.3.1 vorgestellt. Die einzelnen Pfade einer Split/Join-Region bestehen im einfachsten Fall nur aus einer Sequenz, können aber auch aus einer beliebigen Kombination von Kontroll-Regionen aufgebaut sein.
Abbildung 4-6 soll noch einmal den prinzipiellen Aufbau einer Split/Join-Region darstellen und als Grundlage für die folgende Erläuterung der Abschätzung dienen.

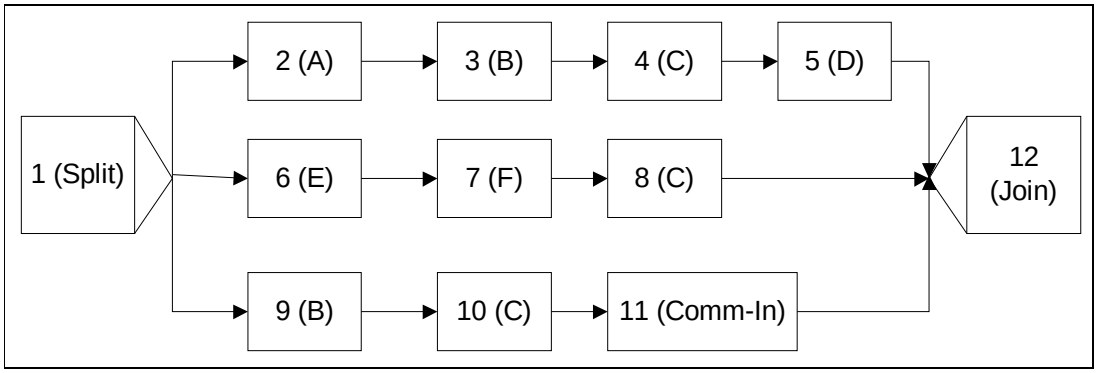


Abb. 4-6: Split/Join-Region.

Bei der Abschätzung einer Split/Join-Region werden zuerst immer alle Pfade betrachtet, egal ob der Split-Knoten ein AND-Split oder ein OR-Split ist. Bei einem AND-Split werden bei der tatsächlichen Ausführung des Workflows auf jeden Fall alle Pfade durchlaufen, bei einem OR-Split nur einige davon. Da aber zum Zeitpunkt der Abschätzung die Bedingungen für die Auswahl der OR-Split-Pfade meistens noch nicht ausgewertet werden können, dürfen auch keine Annahmen darüber gemacht werden, welche Pfade zur Laufzeit tatsächlich ausgeführt werden.
Sollte eine Aussage darüber doch einmal möglich sein, weil die Unterbrechung der Instanz zum Beispiel direkt vor oder am Split-Knoten erfolgte und die Werte zur Auswertung der Split-Bedingungen deshalb schon vorliegen, werden die Pfade einer OR-Verzweigung, die nicht ausgewählt wurden, von der weiteren Abschätzung ausgenommen.
Angenommen in Abbildung 4-6 wäre ein OR-Split dargestellt, von dem bekannt wäre, dass der dritte Pfad nicht ausgewählt wird. Dann sind für die weitere Abschätzung nur die oberen beiden Pfade relevant, der dritte wird nicht weiter betrachtet.

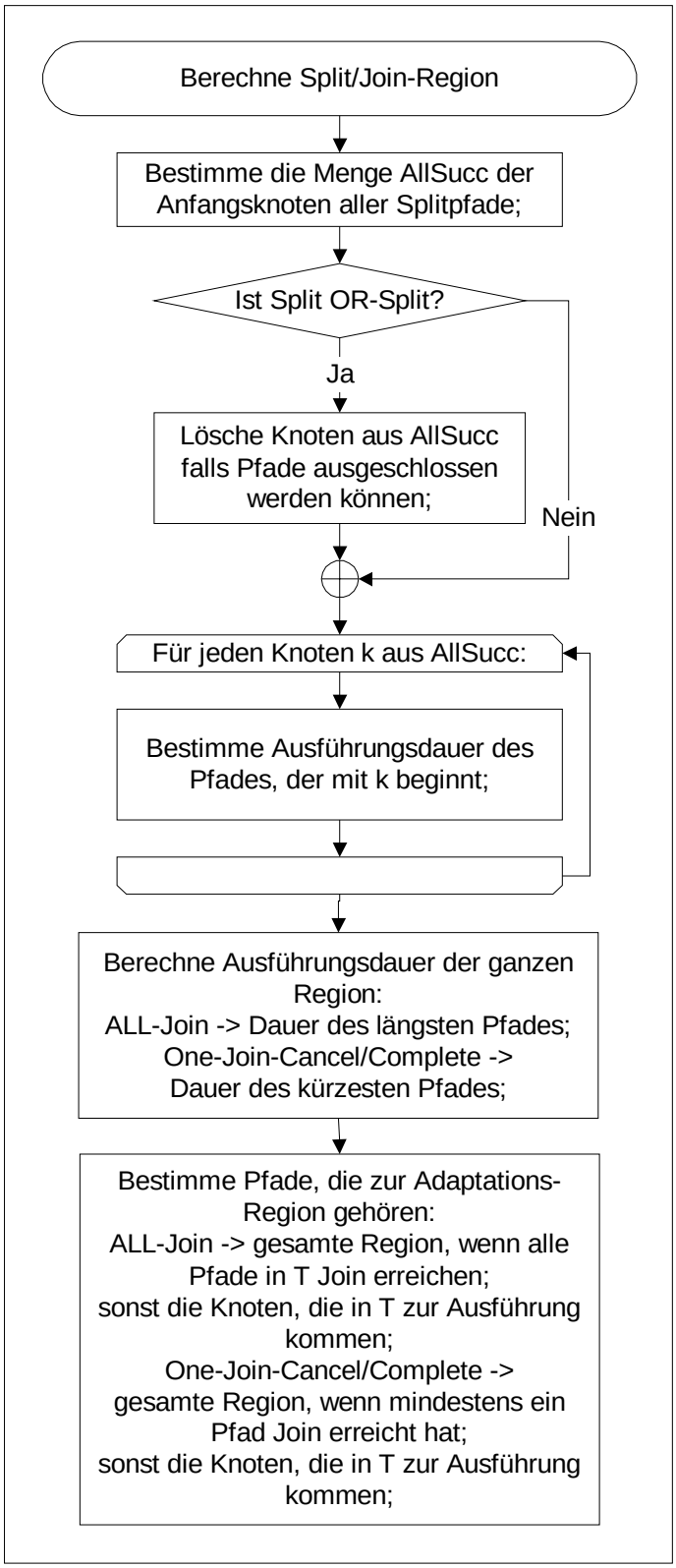


Abb. 4-7: Abschätzung einer Split-Join-Region bei endlichem Gültigkeitsintervall und einer Unterbrechung vor oder am Split-Knoten.

In Abbildung 4-7 ist der Algorithmus für die Abschätzung einer Split/Join-Region für den Fall eines endlichen Gültigkeitsintervalls *T* und einer Unterbrechung vor der Split/Join-Region oder direkt am Split-Knoten dargestellt.
Um die Adaptations-Region AR_T zu bestimmen, wird zuerst die Menge *AllSucc* aller Nachfolger des Split-Knotens bestimmt. Im Beispiel aus Abbildung 4-6 umfasst diese die drei Knoten 2, 6 und 9.
Ist der Split-Knoten ein OR-Split, wird als Nächstes versucht, die konditionalen Transitionen auszuwerten. Gelingt das, können möglicherweise Knoten aus AllSucc gelöscht werden. Danach werden für jeden einzelnen Pfad die Ausführungsdauer und die Knotenmenge bestimmt. Jeder Pfad wird abgelaufen und dabei werden rekursiv die Ausführungsdauern und Knotenmengen der einzelnen Kontroll-Regionen, aus denen er besteht, bestimmt. Die Ausführungsdauern der Kontroll-Regionen werden in der Reihenfolge ihres Auftretens addiert.
Ein Pfad wird solange abgelaufen, bis entweder

- die obere Grenze T_{End} des Gültigkeitsintervalls oder
- der Join-Knoten erreicht wird.

Aus den Ausführungsdauern der einzelnen Pfade wird die Ausführungsdauer der gesamten Split/Join-Region folgendermaßen berechnet:

- Ist der Join-Knoten vom Typ ALL-Join und werden alle Pfade diesen innerhalb des Gültigkeitsintervalls *T* erreichen, ist die Ausführungsdauer der gesamten Split/Join-Region gleich der Ausführungsdauer des längsten Pfades.
- Ist der Join-Knoten vom Typ One-Join-Cancel- oder -Complete und wird mindestens ein Pfad innerhalb *T* den Join-Knoten erreichen, ist die Ausführungsdauer der Split/Join-Region gleich der Ausführungsdauer des kürzesten Pfades.
- Werden bei einem ALL-Join-Knoten nicht alle und bei einem One-Join-Knoten nicht mindestens einer der Pfade diesen in *T* erreichen, ist die Ausführungsdauer der gesamten Split/Join-Region unbekannt und die Adaptations-Region AR_T endet in den einzelnen Pfaden der Region.

Zur Adaptations-Region AR_T gehören dann folgende Knoten:

- Die gesamte Split/Join-Region, wenn bei einem ALL-Join-Knoten alle Pfade den Join-Knoten innerhalb *T* erreichen werden. Erreicht bei einem One-Join-Knoten mindestens ein Pfad innerhalb *T* den Join-Knoten, so gehört dieser Pfad einschließlich des Join-Knotens zu AR_T. Von den übrigen Pfaden gehören die Knoten zu AR_T, die voraussichtlich innerhalb *T* ausgeführt werden. Erreichen alle Pfade innerhalb T den Join-Knoten, so gehört die gesamte Split/Join-Region zur Adaptations-Region.
  Auch bei einem Join-Knoten vom Typ One-Join-Cancel, gehört u. U. die ganze Split/Join-Region zu AR_T. Trifft man während der Abschätzung irgendwelche Annahmen darüber, wo die Pfade abgebrochen werden, wächst die Wahrscheinlichkeit, dass die Adaptations-Region zu klein geschätzt wird und deshalb nicht alle betroffenen Knoten adaptiert werden. Da sowieso alle Pfade untersucht werden, ist kein zusätzlicher Rechenaufwand erforderlich, um alle Knoten, die voraussichtlich in T ausgeführt werden, zu AR_T hinzuzufügen.

- Nur die Knoten der einzelnen Pfade, die voraussichtlich innerhalb des Gültigkeitsintervalls *T* ausgeführt werden, wenn bei einem ALL-Join-Knoten voraussichtlich nicht alle Pfade bzw. bei einem Join-Knoten vom Typ One-Join-Cancel oder One-Join-Complete nicht mindestens ein Pfad diesen innerhalb *T* erreichen werden. Der Join-Knoten gehört auf keinen Fall zur Adaptations-Region, der Split-Knoten schon.

Angenommen der Join-Knoten im obigen Beispiel (Abb. 4-6) wäre vom Typ ALL-Join und nur der letzte Pfad wird diesen innerhalb T erreichen. Im ersten Pfad werden innerhalb *T* nur die Knoten 2 bis 4 und im zweiten Pfad nur die Knoten 6 und 7 ausgeführt werden. Dann ist die Ausführungsdauer für die gesamte Split/Join-Region unbekannt und zur Adaptations-Region AR_T gehören die Knoten 1 bis 4, 6, 7 und 9 bis 11. Die Adaptations-Region endet dann nach den Knoten 4, 7 und 11.

Wurde die Workflow-Instanz unterbrochen, während gerade mehrere Pfade einer Split/Join-Region parallel ausgeführt wurden, müssen die Ausführungsdauer und die Knotenmenge des noch nicht ausgeführten Teils der Split/Join-Region bestimmt werden, der innerhalb des Gül-

tigkeitsintervalls *T* ausgeführt werden wird. Dieser Teil kann bis zum Join-Knoten reichen, er kann aber bei einem sehr kurzen Gültigkeitsintervall oder einer sehr großen Split/Join-Region auch schon vorher enden.

Im in Abbildung 4-8 dargestellten Beispiel wurde die Instanz an den Knoten 3, 7 und 9 unterbrochen. Kann die ganze restliche Split/Join-Region innerhalb *T* ausgeführt werden, gehören die Knoten 3 bis 5 und 7 bis 12 zur Adaptations-Region AR_T. Geht das nicht, weil z. B. der Join-Knoten vom Typ ALL-Join ist und der dritte Pfad diesen nicht innerhalb *T* erreichen wird, gehören nur die Knoten 3 bis 5 und 7 bis 10 zu AR_T. AR_T endet mit den Knoten 5, 8 und 10.

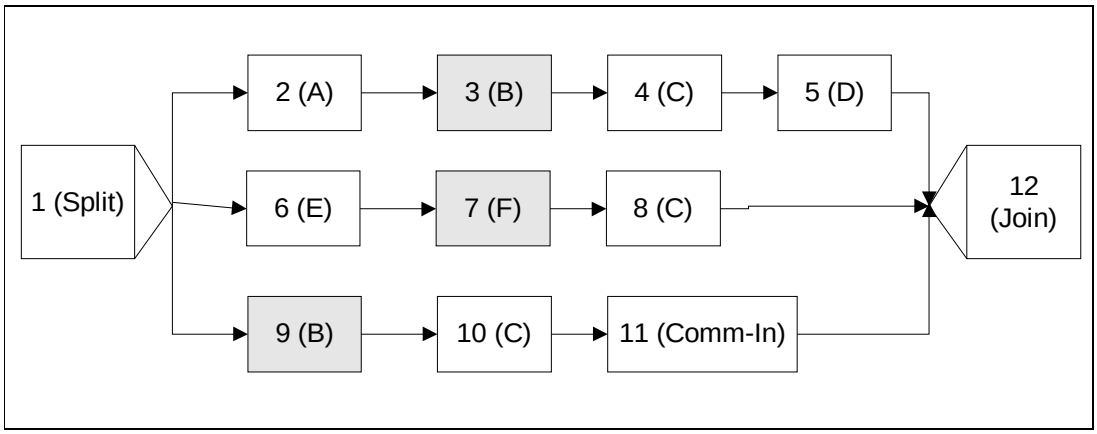


Abb. 4-8: Unterbrechung in einer Split/Join-Region.
(Die Knoten 3, 7 und 9 bilden die Unterbrechungsmenge.)

Bei der Abschätzung der Pfade, die von den Knoten der Unterbrechungsmenge ausgehen, erkennt man in diesem Fall am Auftreten des Join-Knotens, dass man sich in einer Split/Join-Region befindet. Um die Ausführungsdauer der gesamten Kontroll-Region zu berechnen, müssen zunächst die Knoten der Unterbrechungsmenge bestimmt werden, mit denen die übrigen Pfade beginnen.

Angenommen im obigen Beispiel hätte man auf dem ersten Pfad den Join-Knoten erreicht, dann müssen als Nächstes die anderen beiden Pfade, die von den Knoten 7 und 9 ausgehen, untersucht werden.

Hat man für alle Pfade, die sich bei der Unterbrechung in Ausführung befanden, die Ausführungsdauer bis zum Join-Knoten berechnet, bestimmt sich die Ausführungsdauer der gesamten Split/Join-Region analog wie bei einer Unterbrechung vor der Split/Join-Region. Auch die

Knotenmenge der Kontroll-Region, die zur Adaptations-Region AR_T gehört, bestimmt sich analog.

4.2.2.3 Schleife

Mit Schleifen wird die wiederholte Ausführung einer bestimmten Menge von Aktivitäten modelliert. Schleifen haben eine *repeat-until*-Semantik, d. h. die Schleife wird mindestens einmal durchlaufen, die Zahl der weiteren Durchläufe hängt dann von der Schleifenbedingung ab. Solange diese erfüllt ist, wird die Schleife weiter durchlaufen.
In der in Abbildung 4-9 dargestellten Schleife wird die Sequenz der Aktivitäten A, B, C, D einmal durchlaufen. Am Loop-End-Knoten wird überprüft, ob die Schleifenbedingung noch erfüllt ist. Wenn ja, wird die Schleife ein weiteres Mal durchlaufen und am Loop-End-Knoten wird wieder die Bedingung überprüft. Wenn nein, geht der Kontrollfluss zum nächsten Knoten nach dem Loop-End-Knoten über.
Die Knoten zwischen dem Loop-Start- und dem Loop-End-Knoten werden als Schleifenkörper bezeichnet. Dieser kann aus einer oder einer Kombination von mehreren Kontroll-Regionen bestehen. Im Beispiel bilden die Knoten 2 bis 5 den Schleifenkörper.

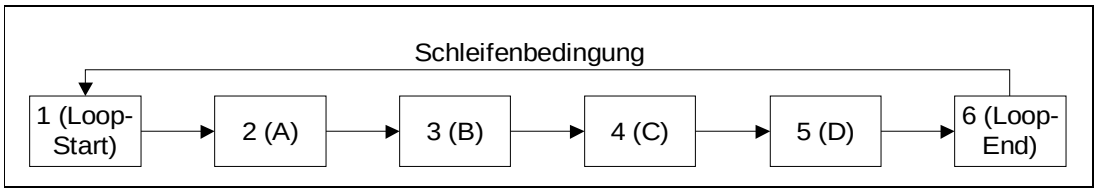


Abb. 4-9: Schleife.

Bei der Abschätzung der Ausführungsdauer einer Schleife sind zwei Faktoren zu berücksichtigen:

- die Ausführungsdauer des Schleifenkörpers und
- die Anzahl der voraussichtlichen Durchläufe.

Die Ausführungsdauer des Schleifenkörpers bestimmt sich durch die Addition der Ausführungsdauern der einzelnen Kontroll-Regionen, aus denen er aufgebaut ist.
Schwieriger zu bestimmen ist die Zahl der Durchläufe. Diese ist immer mindestens eins. Die obere Grenze ist vorher im Allgemeinen nicht bekannt. Für sie könnte man nur einen Schätzwert verwenden, den der Modellierer bei der Definition des Workflows angibt. Um diesen zu verbessern, könnte man analog wie bei den Werten für die Ausführungsdauern einzelner Aktivitäten auch hier während der Ausführung des Workflows mitschneiden, wie oft die Schleife in jeder Instanz tatsächlich durchlaufen wird, und aus diesem Wert statistisch einen Schätzwert für die Durchlaufzahl ermitteln.
Allerdings wird dieser Wert im Allgemeinen nicht den tatsächlichen Wert exakt treffen, da Schleifenbedingungen häufig von lokalen Werten wie z. B. Laborwerten abhängen, die für jeden Patienten zu jedem Zeitpunkt anders sein können.
Bei der Abschätzung der Ausführungsdauer einer Schleife käme also zusätzlich zu der Unexaktheit in den Ausführungsdauern und Wartezeiten der Knoten noch die Unsicherheit über die Anzahl der Durchläufe dazu. Damit wird die Adaptations-Region ungenauer, d. h. die berechnete Adaptations-Region AR_T weicht in mehr Knoten von der tatsächlichen Adaptations-Region ab. Damit wächst die Wahrscheinlichkeit, dass der Workflow erneut unterbrochen werden muss, um Änderungen rückgängig zu machen oder weitere Knoten zu adaptieren. Jede erneute Unterbrechung verzögert aber die Ausführung einer Workflow-Instanz zusätzlich und bedeutet zusätzlichen Rechenaufwand.
Da im Falle einer Schleife die Wahrscheinlichkeit einer erneuten Unterbrechung sehr viel höher ist als bei anderen Kontroll-Regionen, wird beim Auftreten einer Schleife auf die reaktive Adaptations-Strategie (vgl. Abschnitt 3.3) umgeschwenkt. Das wird folgendermaßen umgesetzt:
Wenn während der Abschätzung eines Pfades ein Loop-Start-Knoten auftritt, endet die Adaptations-Region AR_T vor diesem Knoten und die kommenden Knoten werden während der weiteren Ausführung der Instanz überwacht, ob ein zu adaptierender Knoten innerhalb des Gültigkeitsintervalls *T* zur Ausführung kommt.
Trifft man bei der Abschätzung auf einen Loop-End-Knoten, heißt das, dass der Workflow in einer Schleife unterbrochen wurde. In diesem Fall wird überprüft, ob aus den vorliegenden Werten gefolgert werden kann, dass die Schleife sicher nicht mehr durchlaufen wird. Ist das der Fall, gehört der Teil des Schleifenkörpers vom Unterbrechungsknoten bis einschließlich dem

Loop-End-Knoten zur Adaptations-Region AR_T und die Ausführungsdauer der Schleife entspricht der Ausführungsdauer des Teilgraphen vom Unterbrechungsknoten bis zum Loop-End-Knoten. Die Abschätzung des Pfades wird mit dem nächsten Knoten nach dem Loop-End-Knoten fortgesetzt, bis entweder der Endknoten des Workflows oder die obere Grenze T_{End} des Gültigkeitsintervalls erreicht wird.
Ist nicht eindeutig bestimmbar, ob die Schleife weiterhin durchlaufen werden wird, endet die Adaptations-Region AR_T für diesen Pfad am Loop-End-Knoten und nach Wiederanlauf der Instanz wird die weitere Ausführung der Schleife und der Knoten nach der Schleife überwacht, ob im Gültigkeitsintervall *T* ein Knoten zur Ausführung kommt, der zu adaptieren ist.

4.2.2.4 Synchronisations-Transitionen

Mit Synchronisations-Transitionen wird die Reihenfolge zweier Aktivitäten festgelegt, die in parallelen Pfaden liegen. Die Aktivität, von der die Transition ausgeht, muss beendet sein, bevor die Aktivität gestartet wird, bei der die Transition eingeht.
Im in Abbildung 4-10 dargestellten Beispiel muss eine Aktivität vom Typ C beendet sein, bevor eine Aktivität vom Typ D gestartet wird. Im oberen Splitpfad ist das erfüllt, da die Aktivitäten in einer Sequenz liegen, D also auf jeden Fall erst gestartet wird, wenn C zu Ende ist. Da der Aktivitätstyp C im zweiten Pfad nochmals auftritt, muss hier sichergestellt werden, dass der Knoten mit dem Aktivitätstyp D im ersten Pfad erst gestartet wird, wenn auch der Knoten mit dem Aktivitätstyp C im zweiten Pfad beendet ist. Zur Realisierung wird eine Synchronisationskante zwischen Knoten 7 im zweiten Pfad und Knoten 5 im ersten Pfad gezogen. Damit wird der Knoten mit der Aktivität D (5) erst gestartet, wenn beide Knoten mit der Aktivität C (4 und 7) beendet sind.

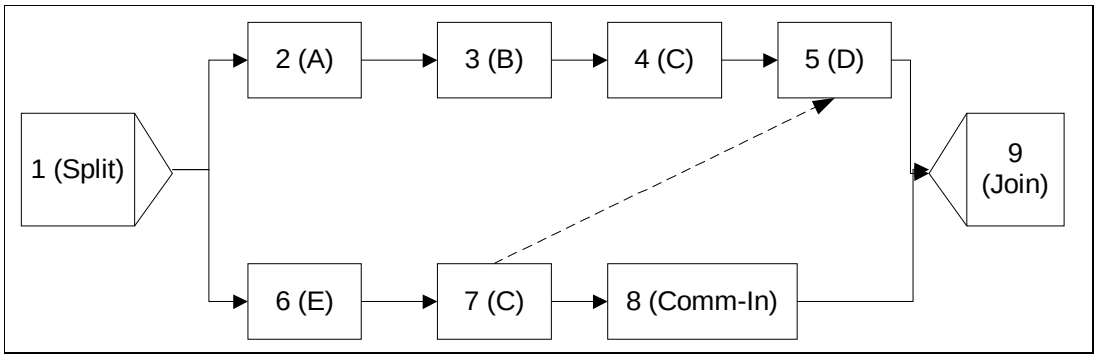


Abb. 4-10: Synchronisations-Transition.

Da Synchronisationskanten den Startzeitpunkt eines Aktivitätsknoten und damit die gesamte Ausführungsdauer eines Pfades beeinflussen, müssen sie bei der Abschätzung der Ausführungsdauer einer Split/Join-Region (nur dort gibt es parallele Pfade zwischen denen Synchronisationskanten gezogen werden können) berücksichtigt werden. Wie das geschieht, ist in Abbildung 4-11 dargestellt und wird im Folgenden beschrieben.

Wenn bei der Abschätzung eines Pfades ein Knoten mit mindestens einer ausgehenden Synchronisations-Transition erreicht wird, wird für diesen Knoten der Wert der Ausführungsdauer des gesamten bisher durchlaufenen Pfades einschließlich des Knotens gespeichert. Danach wird die Abschätzung des Pfades fortgesetzt.
Wird in einem Pfad ein Knoten mit einer oder mehreren eingehenden Synchronisationskanten erreicht, wird zuerst der Wert der Ausführungsdauer des aktuellen Pfades (vor dem Start des Zielknotens der Synchronisationskante) mit den Ausführungsdauern der parallelen Pfade, in denen die Quellknoten der Synchronisations-Transitionen liegen, verglichen. Dabei können folgende drei Fälle auftreten:

- Die Ausführungsdauer des aktuellen Pfades ist größer als alle Ausführungsdauern der Parallelpfade. Die Ausführung der Aktivitätsknoten in den Parallelpfaden ist also schon zu Ende, wenn der zu synchronisierende Knoten gestartet werden soll und der aktuelle Pfad kann normal weiter abgeschätzt werden.

- Die Ausführungsdauer des aktuellen Pfades ist kleiner als die Ausführungsdauer mindestens eines Parallelpfads. Die Ausführung des Quellknotens mindestens einer Synchronisations-Transition ist also noch nicht zu Ende, wenn der Kontrollfluss den zu synchronisierenden Aktivitätsknoten erreicht. Die Ausführungsdauer im aktuellen Pfad muss also auf den größten Wert der Ausführungsdauern der Parallelpfade gesetzt werden. Danach wird der Pfad weiter abgeschätzt.

- Die Ausführungsdauer mindestens eines Parallelpfades ist nicht bekannt. Das kann folgende Ursachen haben:

  - Der Pfad, in dem der Quellknoten der Synchronisations-Transition liegt, wurde noch nicht abgeschätzt. Im obigen Beispiel (vgl. Abb. 4-10) wäre das der Fall, wenn zuerst der erste und dann der zweite Pfad bearbeitet wird. Die Abschätzung des aktuellen Pfades muss vor dem Zielknoten der Synchronisations-Transition abgebrochen werden und kann erst dann fortgesetzt werden, wenn der Parallelpfad abgeschätzt wurde. Dann ist der Vergleichswert für die Ausführungsdauer möglicherweise bekannt.

  - Der Pfad, in dem der Quellknoten der Synchronisations-Transition liegt, wurde schon abgeschätzt und die Adaptations-Region AR_T endet vor dem Quellknoten. Der Quellknoten wird also nicht innerhalb des Gültigkeitsintervalls *T* ausgeführt werden. Folglich kann auch der Zielknoten der Synchronisations-Transition nicht innerhalb *T* ausgeführt werden, da er erst ausgeführt werden darf, wenn der Quellknoten der Synchronisations-Transition beendet ist. Die Adaptations-Region endet im aktuellen Pfad also vor dem zu synchronisierenden Knoten und die Abschätzung dieses Pfades ist beendet.

In den ersten beiden Fällen wird die Abschätzung des Pfades fortgesetzt, bis

- das Ende des Gültigkeitsintervalls *T* oder
- der Join-Knoten der Split/Join-Region oder
- ein weiterer Knoten mit eingehender Synchronisations-Transition erreicht wird.

Bei diesem Vorgehen wird die Wirkung von Synchronisations-Transitionen bei der tatsächlichen Ausführung des Workflows berücksichtigt und die Adaptations-Region AR_T umfasst nicht zu viele Knoten. Das wäre dann der Fall, wenn eingehende Synchronisationskanten nicht berücksichtigt werden würden und deshalb ab dem Zielknoten der Synchronisations-Transition mit einem zu kleinen Wert für die Ausführungsdauer des Pfades gerechnet werden würde.

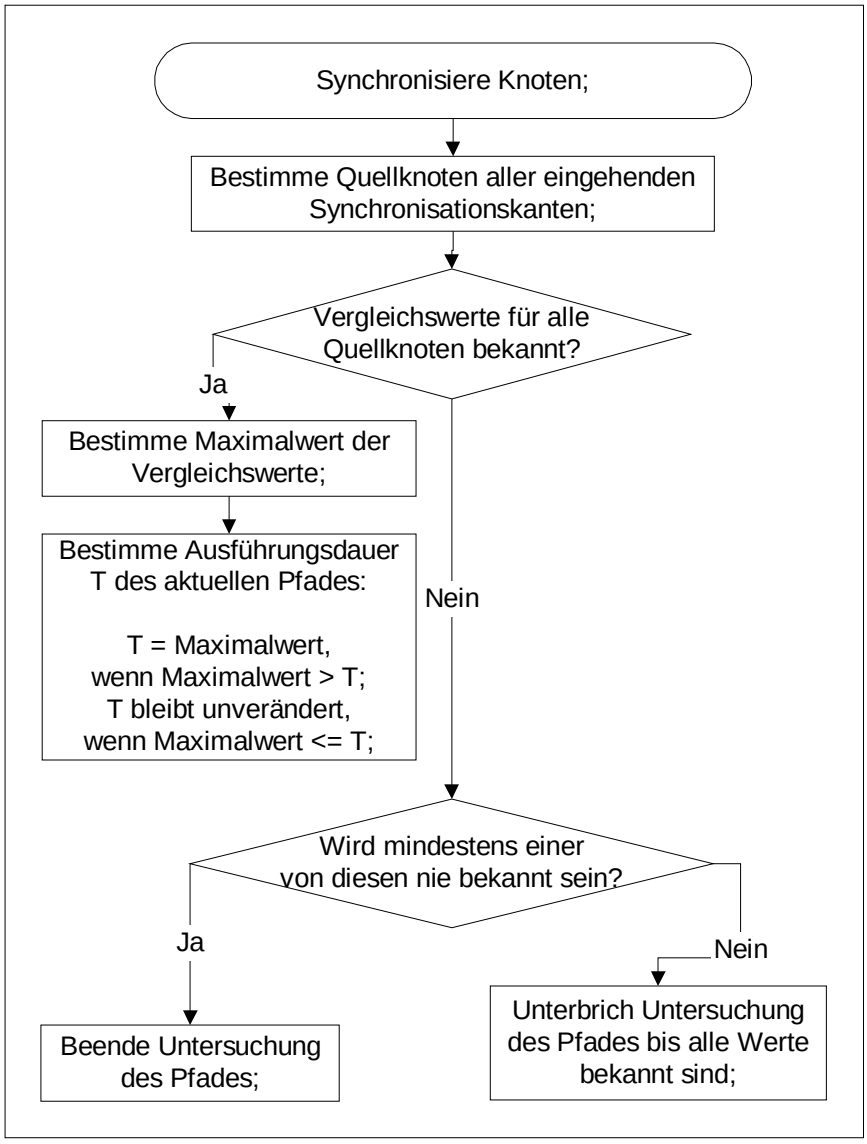


Abb. 4-11: Synchronisation eines Knotens während der Bestimmung der Adaptations-Region.

4.2.2.5 Objektfluss-Transitionen

Objektfluss-Transitionen bilden die Bewegung von Daten im Workflow ab. Ein Aktivitätsknoten hat ausgehende Objektfluss-Transitionen, wenn die Aktivität Daten zur Verfügung stellt, und eingehende Objektfluss-Transitionen, wenn sie Daten benötigt. Kommunikationsknoten können ebenfalls ein- und ausgehende Objektflusskanten haben, da durch sie Daten mit anderen Workflows ausgetauscht werden. Auch konditionale Transitionen können eingehende Objektfluss-Transitionen haben. Für diese wird angenommen, dass die Daten sofort zur Verfügung stehen, wenn der Kontrollfluss die Transition erreicht. Bei der Bestimmung der Adaptations-Region müssen Objektfluss-Transitionen zu konditionalen Transitionen also nicht berücksichtigt werden.
In Abbildung 4-12 ist eine Split/Join-Region mit zwei Objektflusskanten dargestellt. Eine Aktivität vom Typ C benötigt Daten, die eine Aktivität vom Typ E zur Verfügung stellt, d. h. die Knoten 4 und 7 können erst gestartet werden, wenn Knoten 6 beendet ist, da erst dann die Daten zur Verfügung stehen.

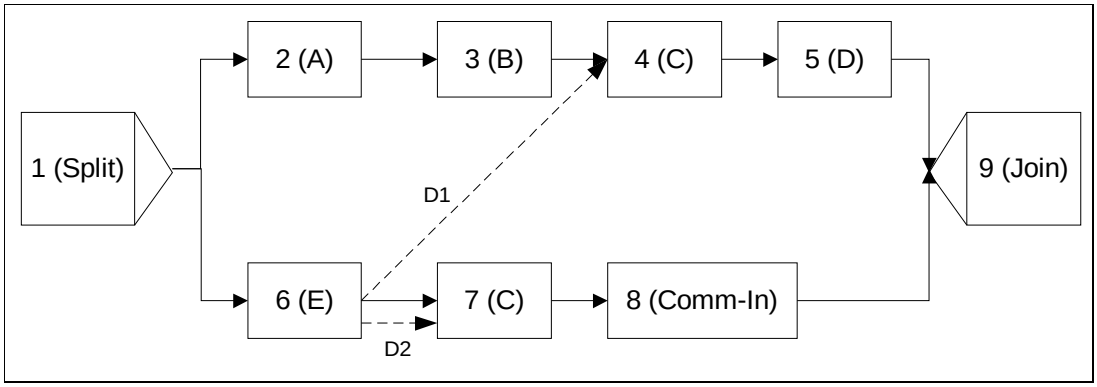


Abb. 4-12: Objektfluss-Transitionen.

Wenn die Knoten wie im Beispiel im unteren Pfad in einer Sequenz hintereinander liegen, muss die Objektfluss-Transition zwischen ihnen bei der Bestimmung der Adaptations-Region AR_T nicht berücksichtigt werden, da der zweite Knoten erst gestartet wird, wenn die Ausführung des ersten zu Ende ist. Liegen die beiden Knoten allerdings in Parallelpfaden (in der Abbildung die Knoten 6 und 4) wirkt die Objektflusskante zwischen ihnen wie eine Synchroni-

sationskante, da der Start der Aktivität, die die Daten benötigt, solange verzögert wird, bis die Daten zur Verfügung stehen.
Bei der Abschätzung der Ausführungsdauer eines Pfades werden Objektfluss-Transitionen deshalb wie Synchronisations-Transitionen behandelt. Die Ausführungsdauern der beiden Pfade werden verglichen und mit dem größeren Wert wird weitergerechnet (vgl. Abschnitt 4.2.2.4).

Zusätzlich muss noch der Fall berücksichtigt werden, dass Daten möglicherweise nicht zur Verfügung stehen, weil

- sie durch technische Fehler verlorengegangen sind oder
- der Pfad, in dem sie erzeugt werden sollten, nicht ausgeführt wurde, da er in einer OR-Verzweigung liegt und nicht ausgewählt wurde.

Bei der Ausführung der Instanz müssen die Daten dann durch eine Nutzereingabe erzeugt werden. Diese Eingabe dauert eine bestimmte Zeit s, die bei der Bestimmung der Ausführungsdauer des Pfades berücksichtigt werden muss. Wenn also während der Bestimmung der Adaptations-Region AR_T schon bekannt ist, dass der Quellknoten der Objektfluss-Transition nie ausgeführt werden wird (weil er z. B. in einem nicht ausgewählten Pfad einer OR-Verzweigung liegt), wird zur Ausführungsdauer des Pfades, in dem der Zielknoten der Objektfluss-Transition liegt, ein bestimmter Wert s für die Dauer der Nutzereingabe addiert.

4.2.3 Ende eines Pfades

Bei den Erläuterungen zur Berechnung der Ausführungsdauern der einzelnen Kontroll-Regionen wurden implizit schon die Bedingungen für den Abbruch der Abschätzung eines Pfades genannt. Im folgenden Abschnitt werden sie noch einmal zusammengefasst.

Für das Ende eines Pfades müssen fünf Fälle unterschieden werden:

- *T* wurde erreicht,
- der Endknoten des Workflows wurde erreicht,

- eine Schleife wurde erreicht,
- ein Knoten konnte nicht synchronisiert werden,
- der Knoten, der die benötigten Daten erzeugt, liegt nicht in der Adaptations-Region AR_T.

Wenn *T* erreicht wurde, heißt das, dass alle Knoten dieses Pfades, die voraussichtlich innerhalb des Gültigkeitsintervalls ausgeführt werden, in der Adaptations-Region AR_T enthalten sind. Der Teilgraph von ART auf diesem Pfad wurde also vollständig bestimmt.
Wenn der Endknoten des Workflows erreicht wurde, wurden ebenfalls alle Knoten auf diesem Pfad, die in AR_T liegen, erfasst. Weiter als bis zum Endknoten kann die Ausführung nicht gehen.
Beim Abbruch vor einer Schleife wurde nur ein Teil der Adaptations-Region AR_T dieses Pfades bestimmt, da die Abschätzung einer Schleife zu ungenau wird, wie in Abschnitt 4.2.2.3 diskutiert wurde. Je geringer die Differenz zwischen der Ausführungsdauer des Pfades bis zur Schleife und *T* ist, desto genauer wurde AR_T bestimmt.
Konnte ein Knoten nicht synchronisiert werden, weil für eine eingehende Synchronisations-Transition der Vergleichswert nicht bekannt war, ist AR_T für diesen Pfad vollständig. Das Fehlen des Vergleichswertes bedeutet, dass der Quellknoten nicht mehr in AR_T liegt. Wenn dieser nicht innerhalb *T* ausgeführt werden kann, kann auch der zu synchronisierende Knoten nicht innerhalb *T* ausgeführt werden, da er erst nach Ende der Ausführung des Quellknotens gestartet werden darf.
Das gleiche gilt für den letzten Fall (fehlende Eingabedaten), auch hier wurde die Adaptations-Region AR_T ganz bestimmt.

4.3 Überwachung des Endes der Adaptations-Region

Damit der Workflow-Monitoring-Agent bei der weiteren Ausführung des adaptierten Workflows weiß, in welchem Teil davon der Adaptations-Agent Änderungen vorgenommen hat, müssen der Anfang und das Ende der Adaptations-Region AR_T für eine bestimmte Kontroll-Aktion gekennzeichnet werden.

Die Adaptations-Region beginnt mit den Knoten der Unterbrechungsmenge und endet entweder mit dem Endknoten des Workflows oder in den einzelnen Pfaden, falls innerhalb eines endlichen Gültigkeitsintervalls *T* nicht der ganze restliche Workflow durchlaufen werden wird.
Die Unterbrechungsmenge und damit der Beginn der Adaptations-Region AR_T sind bekannt und müssen nicht extra gekennzeichnet werden. Beim Ende der Adaptations-Region muss man zwei Fälle unterscheiden:

- Reicht sie bis zum Endknoten des Workflows, muss das Ende nicht zusätzlich gekennzeichnet werden.
- Endet die Adaptations-Region vorher, weil das Gültigkeitsintervall zu Ende ist, ein Knoten nicht synchronisiert werden konnte, eine Schleife erreicht wurde oder Daten nicht innerhalb *T* zur Verfügung stehen werden, muss diese Stelle im Workflow-Graphen markiert werden.

Diese Markierung erfolgt über sogenannte *Monitoring-Knoten.* Sie stellen eine weitere Art von Kontrollknoten dar, d. h. sie haben keine Ausführungsdauer oder Wartezeit und werden in die Transition nach dem letzten Knoten von AR_T eingefügt (vgl. Abbildung 4-13). Trifft die Workflow-Engine bei der weiteren Ausführung der Instanz auf einen solchen Knoten, geht sie einfach zum nächsten Knoten im Kontrollfluss weiter.

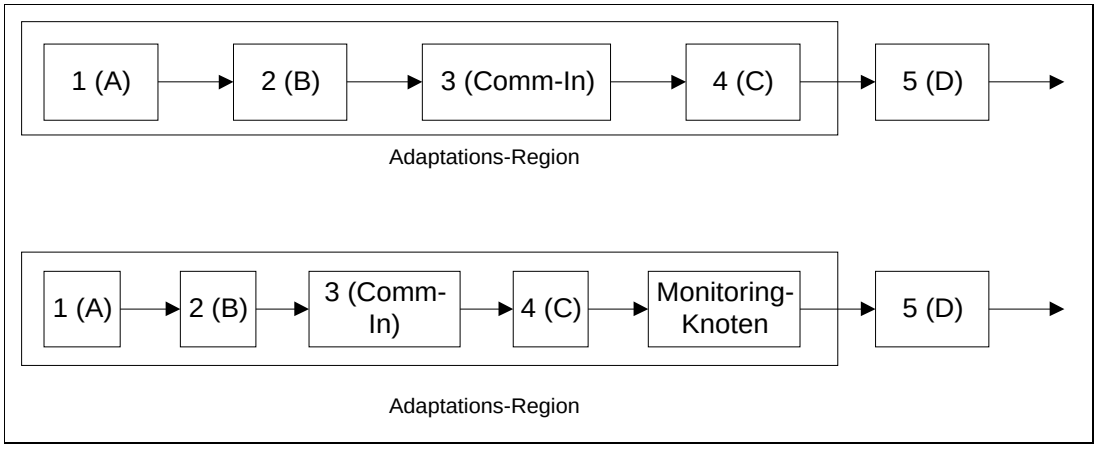


Abb. 4-13: Adaptations-Region vor und nach dem Einfügen eines Monitoring-Knotens.

Man unterscheidet zwei Typen von Monitoring-Knoten:

- Monitoring-Knoten vom Typ-1 kennzeichnen das Ende der Adaptations-Region, wenn die Adaptations-Region für diesen Pfad vollständig bestimmt wurde (vgl. Abschnitt 4.2.3).
- Monitoring-Knoten vom Typ-2 werden verwendet, wenn die Adaptations-Region AR_T wegen des Auftretens einer Schleife auf diesem Pfad nicht vollständig bestimmt wurde (vgl. Abschnitt 4.2.3).

In einer Tabelle der Agenten-Datenbank (vgl. Anhang A) wird gespeichert, zu welcher Kontroll-Aktion ein Monitoring-Knoten gehört.

Es wird nun skizziert, wie der Workflow-Monitoring-Agent bei der Überwachung des Endes der Adaptations-Region AR_T vorgeht.
Wenn die Instanz wieder anläuft, wird für jede Kontroll-Aktion mit endlichem Gültigkeitsintervall *T* der zeitliche Ablauf überwacht. Ist *T* abgelaufen und im Workflow ist noch kein Monitoring-Knoten für diese Kontroll-Aktion aufgetreten, bedeutet das, dass die Adaptations-Region AR_T zu groß geschätzt wurde und somit eventuell Knoten geändert wurden, die nicht mehr innerhalb des Gültigkeitsintervalls zur Ausführung kommen. Für diese Knoten müssen die Änderungen rückgängig gemacht werden. Der Workflow-Monitoring-Agent veranlasst also eine erneute Unterbrechung der Instanz und informiert dann den Adaptations-Agenten, damit dieser die überflüssigen Änderungen rückgängig macht.
Erreicht der Workflow einen Monitoring-Knoten vom Typ-1 obwohl das Gültigkeitsintervall noch nicht vorbei ist, wurde die Adaptations-Region AR_T zu klein geschätzt, d. h. es wurden möglicherweise zu wenig Knoten geändert. Als erstes wird nun überprüft, ob das Intervall vom Zeitpunkt *now,* an dem der Monitoring-Knoten erreicht wurde, bis zur oberen Grenze T_{End} des Gültigkeitsintervalls groß genug ist, um darin noch mindestens die nächste Aktivität des Kontrollflusses auszuführen. Ist das der Fall, wird der Workflow erneut unterbrochen und der Adaptations-Agent aufgerufen, der für die gleiche Kontroll-Aktion mit dem Gültigkeitsintervall (*now,* T_{End}) erneut eine Bestimmung der Adaptations-Region AR_T und Änderungen an darin enthaltenen betroffenen Knoten vornimmt.

Ist das Intervall (*now*, T_{End}) zu klein, um noch die nächste Aktivität ausführen zu können, kann der Workflow weiterlaufen, da keine weiteren betroffenen Knoten innerhalb des Gültigkeitsintervalls zur Ausführung kommen können.
Kommt im Kontrollfluss ein Monitoring-Knoten vom Typ-2 zur Ausführung, wird zur reaktiven Adaptations-Strategie übergegangen. Die weitere Ausführung des Workflows wird bis zum Ende des Gültigkeitsintervalls der Kontroll-Aktion überwacht, ob ein betroffener Knoten zur Ausführung kommt. Ist das der Fall, wird die Instanz unterbrochen und der Adaptations-Agent aufgerufen, um diesen Knoten zu ändern.

4.4 Exaktheit der Abschätzung

Wie schon mehrfach angedeutet, ist die Exaktheit der Abschätzung der Adaptations-Region AR_T ein wichtiges Kriterium für die Arbeit des Agenten. Die Abschätzung ist umso exakter, je näher sie der Wirklichkeit kommt. Wird die Abschätzung zu schlecht, veranlasst das beispielsweise den Agenten von der prädiktiven zur reaktiven Strategie überzugehen. Je genauer die Adaptations-Region den tatsächlich innerhalb des Gültigkeitsintervalls durchlaufenen Teil des Workflows trifft, desto geringer ist die Wahrscheinlichkeit, dass der Workflow erneut unterbrochen werden muss, um Adaptationen nachzuholen oder rückgängig zu machen.

Die Exaktheit der Abschätzung hängt von zwei wichtigen Punkten ab:

- von der Exaktheit der Ausführungsdauern und Wartezeiten der Aktivitäts- und Kommunikationsknoten und
- von der Anzahl der Schleifen in der Adaptations-Region AR_T.

Je weniger Schleifen ein Workflow enthält und je genauer die Werte der Ausführungsdauern und Wartezeiten sind, desto genauer wird die geschätzte Adaptations-Region AR_T mit dem tatsächlich innerhalb *T* durchlaufenen Teil des Workflows übereinstimmen. Im Folgenden werden kurz die Möglichkeiten skizziert, diese beiden Kriterien für die Exaktheit zu beeinflussen.
Wie in Abschnitt 4.2.2.3 ausführlich dargestellt, würde die Abschätzung einer Schleife die Exaktheit der gesamten Abschätzung negativ beeinflussen. Deshalb wird bei Auftreten einer

Schleife die Abschätzung abgebrochen, um die Exaktheit der bisher bestimmten Adaptations-Region AR_T nicht zu verschlechtern.

Die Ausführungsdauer einer Aktivität ist umso exakter, je geringer der Betrag der Differenz zwischen dem tatsächlichen Wert und dem geschätzten ist, der zur Bestimmung der Adaptations-Region verwendet wird. Dieser Schätzwert kann im laufenden Betrieb des Workflow-Systems verbessert werden, indem die realen Werte der Ausführungsdauern mitgeschnitten werden und dann aus diesen mit statistischen Methoden neue Werte für die Ausführungsdauern errechnet werden (vgl. Abschnitt 4.1). Diese werden dann anstelle der bei der Modellierung angegebenen Werte verwendet. Je länger das Gesamtsystem in Betrieb ist, desto mehr reale Werte stehen zur Verfügung und umso genauer bildet der statistische Wert den tatsächlichen Wert ab. Das gleiche Verfahren kann auch für die Wartezeiten der Kommunikationsknoten verwendet werden.

4.5 Behandlung technischer Fehler

Während der Adaptations-Agent die Adaptations-Region AR_T bestimmt und seine Änderungen vornimmt, können technische Fehler auftreten, die seine weitere Arbeit behindern. Das kann z. B. der Ausfall eines Rechners oder die Nichtverfügbarkeit einer Datenbank sein. Es kann auch sein, dass der Agent auf Antworten einer Datenbank sehr lange warten muss, da diese stark frequentiert und deshalb überlastet ist.

Solche Wartezeiten und Fehler verzögern den Wiederanlauf der Instanz signifikant, während im Allgemeinen angenommen wird, dass die Dauer der Abschätzung und der Adaptation eines Workflows im Verhältnis zum kürzesten Gültigkeitsintervall der gefolgerten Kontroll-Aktionen vernachlässigbar ist.

An einem Beispiel soll das verdeutlicht werden (vgl. Abbildung 4-14). Für eine Instanz, deren kürzeste Aktivität drei Minuten dauert, wurde eine Kontroll-Aktion mit einem Gültigkeitsintervall von drei Stunden gefolgert. Die Instanz wird zu Beginn dieser drei Stunden unterbrochen und der Adaptations-Agent bestimmt die Adaptations-Region AR_T für drei Stunden und nimmt darin die Änderungen vor. Wenn keine technischen Fehler oder Verzögerungen auftreten, dauert das z. B. maximal eine Minute, also nur einen Bruchteil des Gültigkeitsintervalls. Wenn die Instanz wieder anläuft, ist nur eine Minute des Gültigkeitsintervalls vorbei, die Adaptations-

Region ist also weiterhin exakt, da innerhalb einer Minute auch die kürzeste Aktivität nicht vollständig ausgeführt werden kann.

Abb. 4-14: Verkürzung des Gültigkeitsintervalls im Normalfall.

Durch Fehler und sehr lange Antwortzeiten kann es während der Adaptation zu einer Verlängerung der Umbauzeit auf 30 Minuten kommen. Wenn die Instanz wieder anläuft, sind also bereits 30 Minuten des Gültigkeitsintervalls von drei Stunden verstrichen (vgl. Abb. 4-15).
Die gefolgerte Kontroll-Aktion gilt also nur noch zweieinhalb Stunden. Der Adaptations-Agent hat aber in dem Teil des Workflows Änderungen vorgenommen, der in den nächsten drei Stunden durchlaufen werden wird. Er hat also mit großer Wahrscheinlichkeit zu viele Knoten geändert, die Instanz muss nach zweieinhalb Stunden erneut unterbrochen werden, um diese Änderungen rückgängig zu machen.

Abb. 4-15: Verkürzung des Gültigkeitsintervalls im Fehlerfall.

Um solche Fehler zu vermeiden, kann der Adaptations-Agent während der Adaptation einer Instanz selbst protokollieren, wie lange er für die Änderungen gebraucht hat. Wenn dieser Zeitraum ein signifikanter Anteil des kürzesten Gültigkeitsintervalls und länger als die Ausführungsdauer der kürzesten Aktivität im Workflow ist, gibt der Agent die Instanz noch nicht frei, sondern versucht erst die geänderten Knoten zu bestimmen, die jetzt schon außerhalb des Gültigkeitsintervalls liegen, um für diese die Änderungen rückgängig zu machen.
Wurde der technische Fehler noch nicht behoben oder ist die Datenbank immer noch überlastet, wird er auch dafür wieder signifikant lange brauchen. Wenn also bemerkt wird, dass die technischen Fehler und Überlastungen weiterhin bestehen, ist es besser, die Instanz wieder anlaufen zu lassen und eine erneute Unterbrechung in Kauf zu nehmen, da bis dahin die technischen Fehler vielleicht behoben sind. Andernfalls kann es sein, dass man mit Ändern und Rücknahme der Änderungen das ganze Gültigkeitsintervall hindurch beschäftigt ist, der Workflow also drei Stunden unterbrochen wäre und die gefolgerten Kontroll-Aktionen nicht berücksichtigt werden würden, was nicht akzeptabel ist.

5. Adaptationen

Nachdem die Adaptations-Region bestimmt wurde, müssen die eigentlichen Adaptationen durchgeführt werden. Dazu werden je nach Kontroll-Aktion zusätzliche Knoten und Transitionen in die unterbrochene Workflow-Instanz eingefügt und vorhandene Knoten und Transitionen gelöscht.
In diesem Kapitel werden zuerst die einzelnen Adaptations-Operationen im Detail vorgestellt. Dabei wird jeweils erläutert, wie der Workflow-Graph umgebaut wird (Abschnitt 5.1).
Danach wird diskutiert, wie die Adaptations-Region durch aufeinanderfolgende Adaptationen beeinflusst wird, wenn für ein Intervall mehrere Kontroll-Aktionen gleichzeitig gelten. Anschließend wird diskutiert, wie diese Beeinflussungen möglichst gering gehalten werden können (Abschnitt 5.2).
Abschließend wird das Zurücksetzen von Adaptationen erläutert. Dieser Schritt ist nötig, wenn die Adaptations-Region AR_T zu groß geschätzt wurde und deshalb Knoten geändert wurden, die dann bei der weiteren Ausführung nicht mehr innerhalb des Gültigkeitsintervalls zur Ausführung kommen. Damit der Workflow weiterhin logisch korrekt bleibt, müssen für diese Knoten die Adaptationen wieder rückgängig gemacht werden (Abschnitt 5.3).

5.1 Adaptation des Kontroll- und Datenflusses

Im folgenden Abschnitt wird zuerst erläutert, wie die von einer Kontroll-Aktion betroffenen Knoten innerhalb einer Adaptations-Region bestimmt werden. Danach wird für jede der Kontroll-Aktionen *check*, *drop*, *replace*, *delay* und *add* dargestellt, welche Änderungen am Kontroll- und Datenfluss vorzunehmen sind.

5.1.1 Bestimmung der betroffenen Knoten

Bevor die Änderungen am Workflow-Graphen für eine bestimmte Kontroll-Aktion vorgenommen werden können, müssen die Knoten gefunden werden, die von den Änderungen betroffen

sind. Das sind diejenigen Aktivitätsknoten, deren Aktivitätstyp mit dem von der Kontroll-Aktion betroffenen übereinstimmt.
Die Menge der betroffenen Knoten muss also bei allen Kontroll-Aktionen außer bei *add*-Kontroll-Aktionen bestimmt werden, da durch diese neue Knoten eingefügt und keine vorhandenen Knoten geändert werden.

Die Menge dieser Knoten bestimmt sich unterschiedlich je nachdem, ob die reaktive oder die prädiktive Adaptations-Strategie (vgl. Abschnitt 3.3) angewendet wird.
Bei der reaktiven Strategie wird immer dann adaptiert, wenn ein von einer gültigen Kontroll-Aktion betroffener Knoten zur Ausführung kommt. Werden mehrere Pfade parallel ausgeführt, können auch mehrere betroffene Knoten gleichzeitig zur Ausführung kommen. Sie sind bekannt und werden dem Adaptations-Agenten explizit übergeben.
Wird die prädiktive Strategie angewendet, gibt es zu jeder Kontroll-Aktion eine aktuelle Adaptations-Region AR_T, in der betroffene Knoten liegen können. Um diese Knoten zu finden, muss die gesamte aktuelle Adaptations-Region durchsucht werden, ob darin Aktivitätsknoten liegen, denen die von der Kontroll-Aktion betroffene Aktivität zugeordnet ist.
Die so bestimmte Menge kann auch leer sein, denn bei der prädiktiven Strategie wird dann unterbrochen, wenn eine Kontroll-Aktionen aufgrund eines oder mehrerer Ereignisse gefolgert wurde, und nicht erst, wenn ein betroffener Knoten vor der Ausführung steht. Ist das Gültigkeitsintervall der Kontroll-Aktion kurz, kann es sein, dass keine betroffene Aktivität darin zur Ausführung kommt.

5.1.2 Kontroll-Aktion *check*(A)

Eine *check*-Kontroll-Aktion bedeutet, dass die Aktivität möglicherweise nicht mehr adäquat ist aber aus der zur Verfügung stehenden Wissensbasis keine der anderen Kontroll-Aktionen eindeutig gefolgert werden konnte.
Die Aufgabe des Adaptations-Agenten ist es, alle betroffenen Knoten aus der Adaptations-Region AR_T über einen Dialog dem Nutzer anzuzeigen. Dieser kann dann entscheiden, ob ein Knoten unverändert bleiben soll oder ob eine der Kontroll-Aktionen *drop, replace* oder *delay* anzuwenden ist. Eine erneute *check*-Kontroll-Aktion wäre unsinnig, da sie nur eine Wiederho-

lung darstellen würde. Eine *add*-Kontroll-Aktion kann ebenfalls nicht angegeben werden, da sie sich nicht auf schon in AR_T enthaltene Aktivitätsknoten bezieht sondern auf neu einzufügende. Damit gibt sie nicht an, wie die inadäquate Aktivität zu behandeln ist. Möchte der Nutzer dennoch eine zusätzliche Aktivität ausführen lassen, kann er bei der abschließenden Überarbeitung der an der Workflow-Instanz vorgenommenen Änderungen einen zusätzlichen Aktivitätsknoten einfügen.
Hat der Nutzer sich für eine neue Kontroll-Aktion entschieden, so wird diese in die Liste der zu bearbeitenden Kontroll-Aktionen eingefügt. Soll ein Knoten unverändert bleiben, muss der Adaptations-Agent keine weiteren Aktionen für diesen Knoten durchführen.

5.1.3 Kontroll-Aktion *drop*(A)

Eine *drop*-Kontroll-Aktion bewirkt, dass alle von ihr betroffenen Knoten aus der Adaptations-Region AR_T gelöscht werden. Dabei werden für jeden betroffenen Knoten die folgenden Schritte durchgeführt:

- Umlenken des Datenflusses,
- Löschen des Knotens und seiner ein- und ausgehenden Kanten,
- Einfügen einer neuen Transition.

Als Erstes muss der eventuell vorhandene Datenfluss umgelenkt werden, damit keine Daten verloren gehen und auch nach Löschen des Knotens alle übrigen Aktivitäten ihre Daten bekommen. Dabei ist zwischen internem (d. h. zwischen Knoten bzw. zwischen Knoten und Transitionen) und externem (d. h. aus einer bzw. in eine Datenbank) Datenfluss zu unterscheiden.
Erhält der zu löschende Knoten seine Eingabedaten aus einer Datenbank, kann die eingehende Datenflusskante einfach gelöscht werden, da es keinen Einfluss auf den übrigen Datenfluss hat, ob die Daten ausgelesen werden oder nicht.
Genauso verhält es sich, wenn die Ausgabedaten in eine Datenbank ausgeschrieben werden. Liest eine Aktivität später diese Daten aus, erhält sie eine ältere Version, als die, die der gelöschte Knoten produziert hätte, oder die Daten sind gar nicht vorhanden. Dann müssen sie

von der Workflow-Engine oder der Aktivität selbst beschafft werden (z. B. über eine Nutzereingabe). Das Risiko veralteter Daten bzw. einer zusätzlichen Nutzereingabe muss aber in Kauf genommen werden, da sonst keine Daten-produzierenden Knoten gelöscht werden dürften, was eine zu strikte Einschränkung wäre.

Beim internen Datenfluss dürfen die Transitionen nicht einfach gelöscht werden, da dadurch Daten verloren gehen können oder andere Aktivitäten keine Daten zur Verfügung gestellt bekommen. In Abbildung 5-1 ist ein Beispiel für internen Datenfluss dargestellt. Der zu löschende Knoten 2 hat eine ein- und eine ausgehende interne Datenflusskante (gestrichelt dargestellt). Wenn dieser Knoten einfach entfernt wird, ohne diese Transitionen zu berücksichtigen, gehen zum einen die Daten verloren, die Knoten 1 produziert, da sie nirgends gespeichert oder verarbeitet werden. Zum anderen stehen Knoten 3 seine benötigten Daten nicht zur Verfügung, da der Knoten, der sie produziert hätte, nicht mehr existiert.

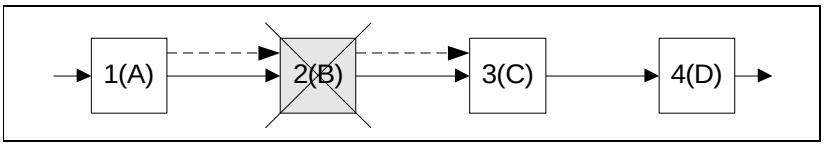


Abb. 5-1: Interner Datenfluss.

Zur Vermeidung solcher Situationen werden die eingehenden internen Datenflusskanten eines zu löschenden Knoten in eine Datenbank umgeleitet, damit die Daten auf keinen Fall verloren gehen. Bei ausgehenden internen Datenflusskanten wird zuerst versucht, einen anderen Knoten zu finden, der die gleichen Daten produziert, im Kontrollfluss vor dem Zielknoten der Datenfluss-Transition liegt und sich noch im Zustand Untouched befindet. Gibt es einen solchen Knoten, kann er als Quellknoten der Datenflusskante eingesetzt werden. Findet sich kein solcher Knoten, müssen die erforderlichen Werte durch eine Datenbankanfrage generiert werden. Diese Lösung ist zusammen mit der Umleitung der eingehenden Datenflusskante in Abbildung 5-2 dargestellt.

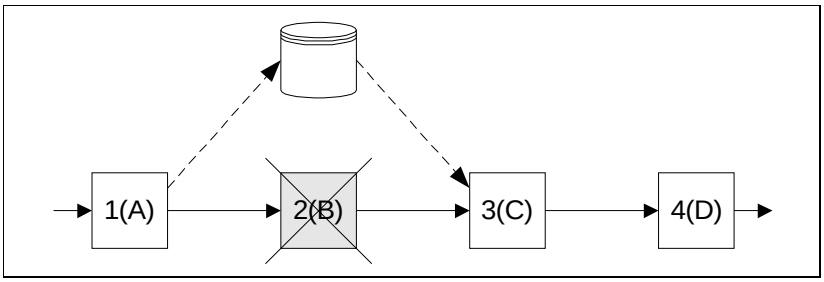


Abb. 5-2: Umleitung des internen Datenflusses über eine Datenbank.

Ist so sichergestellt, dass keine Daten verloren gehen und alle Knoten, die im Kontrollfluss nach dem zu löschenden Knoten liegen, ihre Daten bekommen, kann der Knoten aus der Workflow-Instanz gelöscht werden. Zusätzlich sind auch seine ein- und ausgehende Kontrollfluss-Transitionen und eventuell vorhandene Synchronisations-Transitionen zu löschen. Kontrollflusskanten können ohne weiteres gelöscht werden, da sie ohne den Knoten, zu dem sie führen oder von dem sie kommen, ohne Bedeutung sind. Das gilt auch für Synchronisationskanten, da eine Synchronisation mit einem nicht vorhandenen Knoten sinnlos ist.
Abbildung 5-3 zeigt den obigen Ausschnitt des Workflow-Graphen nach Löschen des Knotens 2 und seiner ein- und ausgehenden Kontrollfluss-Transitionen.

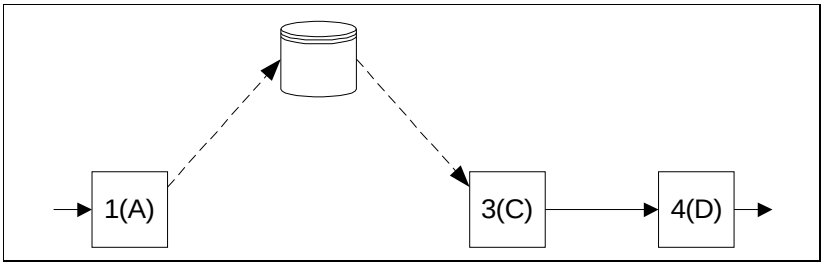


Abb. 5-3: Workflow-Ausschnitt nach dem Löschen von Knoten und Kanten.

Damit der Kontrollfluss nicht unterbrochen ist, sondern vom Vorgänger- zum Nachfolgerknoten weiterlaufen kann, muss noch eine neue Transition zwischen diesen beiden Knoten eingefügt werden. Dabei ist darauf zu achten, ob es zwischen den Aktivitäten dieser Knoten temporale Abhängigkeiten der Form *„zwischen Aktivität A und Aktivität C müssen mindestens drei Stunden liegen“* gibt (bezogen auf das Beispiel). Diese sind in einer Transitions-Bedin-

gung auszudrücken. Wenn einer oder beide der Knoten Kontroll- oder Kommunikationsknoten sind, gibt es solche Abhängigkeiten im Allgemeinen nicht. Der Adaptations-Agent kann solche Abhängigkeiten nur berücksichtigen, wenn sie in der Wissensbasis abgelegt sind, ansonsten muss sie der Nutzer bei der abschließenden Bearbeitung der Änderungen eintragen.
Das Ergebnis der *drop*-Kontroll-Aktion für den Knoten 2 des Beispiels ist in Abbildung 5-4 dargestellt. Die neue Kontrollfluss-Transition ist darin fett gezeichnet.

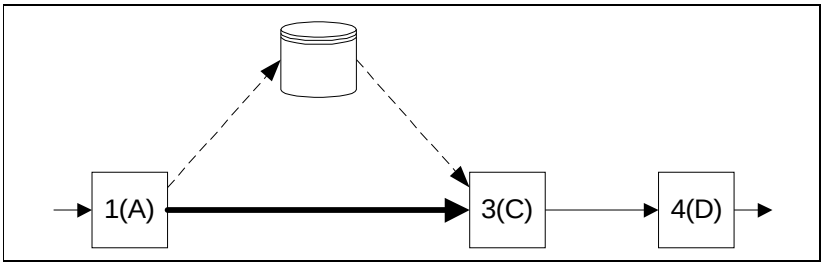


Abb. 5-4: Workflow-Ausschnitt nach Anwendung einer *drop*-Kontroll-Aktion.

Bei der Anwendung einer *drop*-Kontroll-Aktion ist ein Sonderfall zu beachten: Wird aus einer Split/Join-Region mit zwei parallelen Pfaden, von denen einer nur einen Aktivitätsknoten enthält, dieser Knoten gelöscht, sind auch Split- und Join-Knoten zu löschen.
Abbildung 5-5 verdeutlicht die Situation. Der Datenfluss wird hier nicht näher betrachtet, er ist genauso wie im allgemeinen Fall vor dem Löschen eventuell umzuleiten.

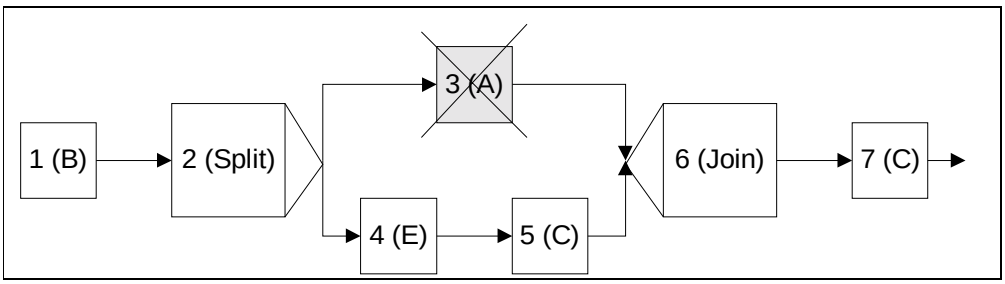


Abb. 5-5: Löschen des letzten Knotens eines Parallelpfades.

Wenn nur Knoten 3 gelöscht werden würde, bliebe eine Split/Join-Region mit nur einem Pfad übrig, der zweite bestände nur aus einer Transition. Da das der Semantik einer solchen Kontroll-Region widerspricht, parallel auszuführende Aktivitäten zu modellieren, müssen Split- und Join-Knoten ebenfalls gelöscht werden. Damit wird die durch das Löschen neu entstandene Semantik (eine Menge von Knoten soll in einer Sequenz ausgeführt werden) auch in den Kontroll-Strukturen korrekt abgebildet.
Das Ergebnis der Löschoperation in diesem Spezialfall wird in Abbildung 5-6 illustriert.

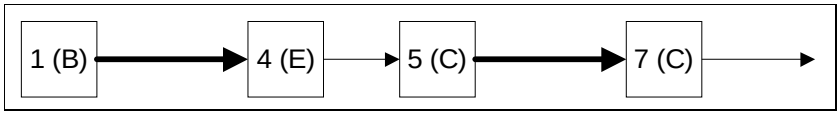


Abb. 5-6: Workflow-Ausschnitt nach dem Löschen des vorletzten Pfades einer Split/Join-Region.

Die beiden fett gezeichneten Transitionen wurden neu eingefügt, damit der Kontrollfluss dieser Sequenz korrekt ist. Wie beim Löschen aus einer Sequenz ist auch hier zu überprüfen, ob eventuell temporale Abhängigkeiten zwischen den im Kontrollfluss jetzt direkt aufeinander folgenden Aktivitäten bestehen, die in Form einer Transitions-Bedingung berücksichtigt werden müssen.

5.1.4 Kontroll-Aktion *replace*(A, B)

Ist eine *replace*-Kontroll-Aktion zu bearbeiten, muss für jeden betroffenen Knoten der ihm zugeordnete Aktivitätstyp A durch den neuen Aktivitätstyp B ersetzt werden. Dazu sind die folgenden Schritte notwendig:

- Ersetzen des alten Aktivitätstyps des Knotens durch den neuen,
- Überarbeiten der Kontrollfluss-Transitionen des Knotens,
- Überarbeiten der Objektfluss-Transitionen des Knotens,
- Überarbeiten der Synchronisations-Transitionen des Knotens.

Als Erstes wird dem Knoten ein neuer Aktivitätstyp zugeordnet. Dabei sind noch keine Änderungen am Kontroll- oder Datenfluss nötig.
In Abbildung 5-7 ist ein Ausschnitt aus einem Workflow-Graphen dargestellt, für den die Kontroll-Aktion *replace(B, E)* durchgeführt werden soll. In der oberen Hälfte ist die ursprüngliche Sequenz abgebildet, die untere Hälfte zeigt diese nach dem ersten Schritt, der Ersetzung des Aktivitätstyps des Knotens.

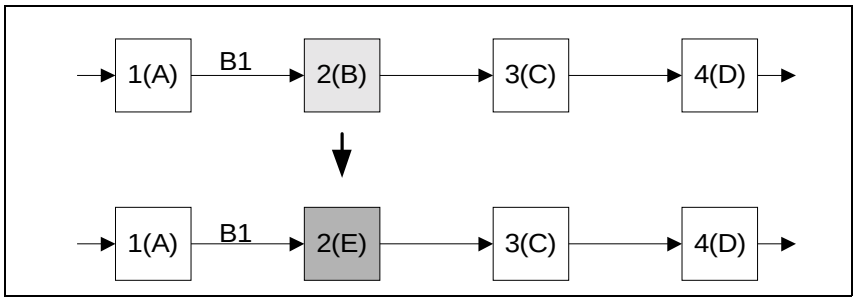


Abb. 5-7: Erster Schritt der Kontroll-Aktion *replace(B, E)*.

Als nächstes werden die ein- und ausgehenden Kontrollfluss-Transitionen des geänderten Knotens überprüft. Dabei ist zuerst zu berücksichtigen, ob diese bereits eine Transitions-Bedingung haben und wenn ja, ob diese auch für die Kombination des Vorgänger- bzw. Nachfolgerknotens mit dem neuen Aktivitätstyp gilt. Ist das der Fall, bleibt die Kante unverändert. Wenn nicht, wird die alte Bedingung gelöscht. Dann wird über die Wissensbasis ermittelt, ob es zusätzliche Bedingungen gibt, die berücksichtigt werden müssen. Diese werden dann der entsprechenden Transition zugeordnet.
Im Beispiel hat die eingehende Kante des betroffenen Knotens 2 die Bedingung B1. Angenommen, diese gilt nur für die Kombination *Aktivitätstyp A - Aktivitätstyp B* aber nicht für *Aktivitätstyp A - Aktivitätstyp E*. Dann müsste diese Bedingung gelöscht werden. Zusätzlich gäbe es aber eine Regel, die vorschreibt, dass zwischen den Aktivitätstypen E und C eine bestimmte Zeit gewartet werden muss. Der ausgehenden Transition des betroffenen Knoten muss also eine neue Bedingung B2 zugeordnet werden. Das Ergebnis dieser Operationen wird in Abbildung 5-8 gezeigt.

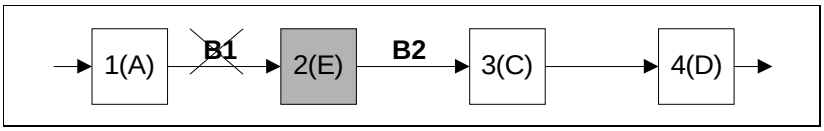


Abb. 5-8: Löschen und Neusetzen von Transitions-Bedingungen bei der Bearbeitung einer *replace*-Kontroll-Aktion.

Im dritten Schritt sind die ursprüngliche und die neue Aktivität zu vergleichen, ob sie die gleichen Daten benötigen und erzeugen. Ist das der Fall, können die bestehenden Objektflusskanten unverändert bleiben.

Benötigt der neue Aktivitätstyp andere Eingabedaten als der alte, muss zunächst untersucht werden, ob es im Workflow einen Aktivitätsknoten gibt, der diese Daten produziert, im Kontrollfluss vor dem geänderten Aktivitätsknoten liegt und noch im Zustand Untouched ist. Gibt es einen solchen Knoten und hat der zu ändernde Knoten bereits eine eingehende interne Datenflusskante, wird ihr als neuer Quellknoten der oben gefundene Knoten zugewiesen. Ist die bestehende eingehende Datenflusskante des zu ändernden Knotens eine externe Datenflusskante, wird sie in eine interne Datenflusskante mit dem oben gefunden Knoten als Quellknoten umgewandelt. Benötigte der alte Aktivitätstyp keine Daten, existiert also weder eine interne noch eine externe Datenflusskante, wird zwischen dem Daten-erzeugenden und dem zu ändernden Knoten eine neue interne Datenfluss-Transition gezogen.

Gibt es keinen Aktivitätsknoten, der die passenden Daten produziert, müssen die von dem neuen Aktivitätstyp benötigten Daten aus einer Datenbank eingelesen werden. Gehörte die bestehende eingehende Kante also vorher zum internen Datenfluss, muss sie in eine externe Datenflusskante umgewandelt werden, d. h. die eingehenden Daten werden aus einer Datenbank ausgelesen. Eine schon existierende externe Datenflusskante muss entsprechend abgeändert werden, und falls noch keine Datenflusskante existiert, wird eine neue eingehende externe Datenflusskante eingefügt.

Unterscheiden sich die Ausgabedaten des alten und des neuen Aktivitätstyps und existiert bereits eine externe Datenflusskante, so ist diese so abzuändern, dass die Daten, die der geänderte Aktivitätsknoten produziert, in der richtigen Datenbank an der richtigen Stelle gespeichert werden. Ist die ausgehende Datenflusskante eine interne Datenflusskante, so ist sie erstens in eine externe Datenfluss-Transition umzuwandeln, damit die Daten, die die neue

Aktivität produziert, nicht verloren gehen. Zweitens ist dafür zu sorgen, dass der Zielknoten der ursprünglichen internen Datenflusskante die benötigten Eingabedaten auch weiterhin bekommt. Dazu ist entweder ein Knoten zu finden, der diese Daten erzeugt, im Kontrollfluss vor diesem Knoten liegt und noch im Zustand Untouched ist, so dass er als Quellknoten dienen kann, oder die Daten sind über eine externe Datenflusskante einzulesen.

Die Abbildung 5-9 zeigt folgende Situation: Der ursprüngliche Aktivitätstyp B benötigt die Daten D1, die über eine externe Datenflusskante aus einer Datenbank eingelesen werden, und produziert die Daten D2, die über eine interne Datenflusskante an die Aktivität vom Typ D weitergeleitet werden. Der neue Aktivitätstyp E benötigt die Daten D3 und produziert die Daten D4. Die vorhandenen Datenfluss-Transitionen müssen also folgendermaßen angepasst werden: Die eingehende externe Datenflusskante kann in eine interne Datenflusskante umgewandelt werden, da der Aktivitätstyp A, der vor dem neuen Aktivitätstyp E liegt, die benötigten Daten D3 produziert. Die ausgehende interne Datenflusskante muss in zwei externe Transitionen aufgespalten werden. Zum einen werden die von der Aktivität vom Typ E produzierten Daten D4 in eine Datenbank ausgeschrieben, damit sie nicht verloren gehen, zum anderen muss die interne Datenflusskante, die die Daten D2 vorher an die Aktivität vom Typ D weitergeleitet hat, in eine externe Datenfluss-Transition umgewandelt werden, da es keinen passenden Knoten gibt, der diese Daten zur Verfügung stellt.

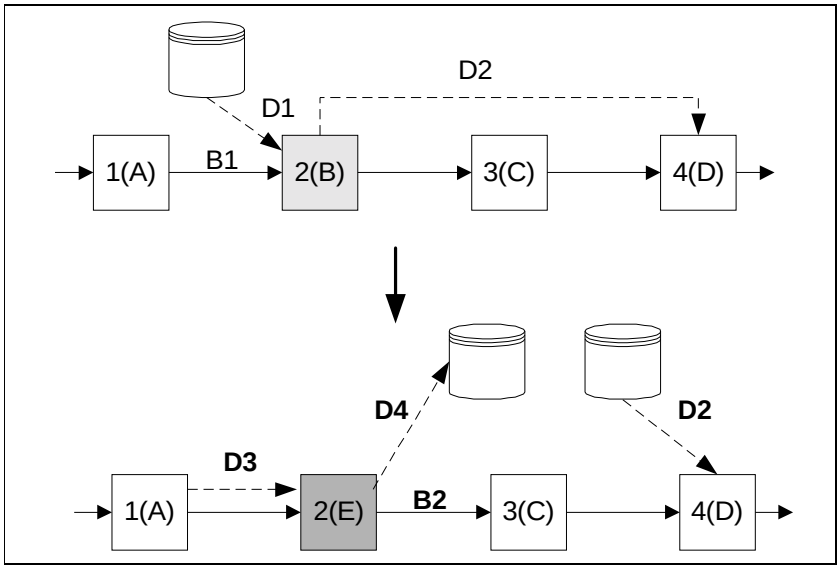


Abb. 5-9: Anpassung der Datenflusskanten bei der Bearbeitung einer *replace*-Kontroll-Aktion.

Als Letztes müssen noch Synchronisations-Transitionen berücksichtigt werden, falls der betroffene Knoten in einer Split/Join-Region liegt. Zuerst sind vorhandene ein- und ausgehende Synchronisationskanten zu untersuchen, ob sie noch benötigt werden, d. h. ob der neue Aktivitätstyp mit den gleichen Aktivitätstypen synchronisiert werden muss wie der alte. Ist das der Fall, können die bestehenden Kanten unverändert bleiben. Sollte eine Synchronisation mit einigen Aktivitätstypen nicht mehr nötig sein, können die entsprechenden Kanten gelöscht werden. Danach muss überprüft werden, ob neue Synchronisations-Transitionen einzufügen sind, weil der neue Aktivitätstyp mit anderen Aktivitätstypen synchronisiert werden muss als der alte.

Auch dieses Vorgehen soll an einem Beispiel erläutert werden (siehe Abb. 5-10). Angenommen die Aktivität vom Typ B wurde durch die Aktivität vom Typ E ersetzt. Dadurch ist die Synchronisationskante von Knoten 4 zu Knoten 2 überflüssig geworden, da der Aktivitätstyp A zwar mit Typ B nicht aber mit Typ E zu synchronisieren ist. Sie kann gelöscht werden. Zusätzlich ist eine neue Synchronisations-Transition zwischen Knoten 2 und Knoten 5 einzufügen, da die Aktivität vom Typ C erst gestartet werden darf, wenn die Aktivität vom Typ E zu Ende ist.

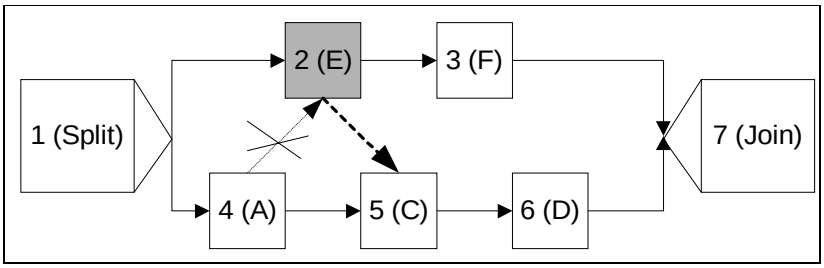


Abb. 5-10: Überarbeitung von Synchronisations-Transitionen bei der Bearbeitung einer *replace*-Kontroll-Aktion.

Die neu eingefügte Synchronisations-Transition ist fett gezeichnet, die zu löschende durchkreuzt. Mit der Untersuchung der Synchronisations-Transitionen ist die Bearbeitung einer *replace*-Kontroll-Aktion abgeschlossen.

5.1.5 Kontroll-Aktion *delay*(A, t)

Bei einer *delay*-Kontroll-Aktion ist die Ausführung aller betroffenen Knoten in der Adaptations-Region um die Zeit *t* zu verzögern. Dabei muss zunächst entschieden werden, auf welche Art die Verzögerung erfolgen soll.
Die Verzögerung erfolgt durch Verschieben des zu verzögernden Knotens von seiner ursprünglichen Position um k Knoten nach hinten, wenn

- in der Zeit *t* genau die nächsten k Knoten ausgeführt werden können,
- es keine Regel gibt, die die Umsortierung der Knoten verbietet und
- der zu verzögernde Knoten keine ausgehende Datenflusskante hat, die zu einem Knoten führt, der nach der Umsortierung vor dem zu verzögernden Knoten liegen würde.

Andernfalls (d. h. es können nicht genau k Knoten in *t* ausgeführt werden oder die Knoten dürfen nicht umsortiert werden) wird zur Verzögerung eine Split/Join-Region mit AND-Split- und ALL-Join-Knoten konstruiert.
Je nach Verzögerungsart sind unterschiedliche Adaptationen am Kontrollfluss der betroffenen Workflow-Instanz vorzunehmen. Diese werden im Folgenden erläutert.

Erfolgt die Verzögerung durch eine Umsortierung der Knoten, sind folgende Schritte nötig:

- Verschieben des Knotens an seine neue Position,
- Überprüfung der Transitions-Bedingungen,
- Überprüfung eingehender interner Objektfluss-Transitionen, ob an der neuen Position aktuellere Daten zur Verfügung stehen.

Um den Knoten an seine neue Position zu verschieben, müssen seine ein- und ausgehenden Kontrollfluss-Transitionen, und die Transition, in die er eingefügt werden soll, folgendermaßen umgesetzt werden:

- Die eingehende Kontrollfluss-Transition führt vom neuen Vorgänger des zu verzögernden Knotens zu dem Knoten selbst.

- Die ausgehende Kontrollfluss-Transition führt vom ehemaligen Vorgänger zum ehemaligen Nachfolger des zu verzögernden Knotens.
- Die Transition, in die der Knoten eingefügt wird, wird zur neuen ausgehenden Kontrollfluss-Transition des zu verzögernden Knotens.

Anschließend wird zuerst überprüft, ob sich durch die neue Reihenfolge der Aktivitäten neue Transitions-Bedingungen ergeben oder vorhandene überflüssig geworden sind. Dann werden noch die eingehenden internen Objektflusskanten des zu verzögernden Knotens überprüft (falls vorhanden). Erzeugt einer der k Knoten, die jetzt vor dem zu verzögernden Knoten ausgeführt werden, die passenden Daten, so kann dieser als Quellknoten der eingehenden Objektfluss-Transition verwendet werden, da seine Daten neuer sind, als die des ursprünglichen Quellknotens. Bei externen eingehenden Objektfluss-Transitionen sind solche Überlegungen nicht nötig, da die Daten erst dann aus der Datenbank eingelesen werden, wenn sie benötigt werden. Es steht also immer die aktuellste Version zur Verfügung.
Abbildung 5-11 zeigt einen Ausschnitt aus einem Workflow-Graphen, in dem die Aktivität vom Typ B um *t* verzögert werden soll. Angenommen in der Zeit *t* könnten die beiden Knoten 3 und 4 ausgeführt werden und es gibt keine Regel, die die neue Reihenfolge verbietet. Dann werden die Knoten in die Reihenfolge 1, 3, 4, 2 gebracht. Danach werden die Transitions-Bedingungen überprüft. Angenommen zwischen der Aktivität A und der Aktivität C muss immer eine Stunde liegen. Dann müsste der neuen Transition zwischen den Knoten 1 und 3 die Transitions-Bedingung *wait 1 hour* zugeordnet werden. Außerdem benötigt die Aktivität B die Daten d, die ihr eine Aktivität vom Typ C zur Verfügung stellt. Einer eingehenden internen Datenflusskante (gestrichelt gezeichnet) könnte also Knoten 3 als neuer Quellknoten zugeordnet werden

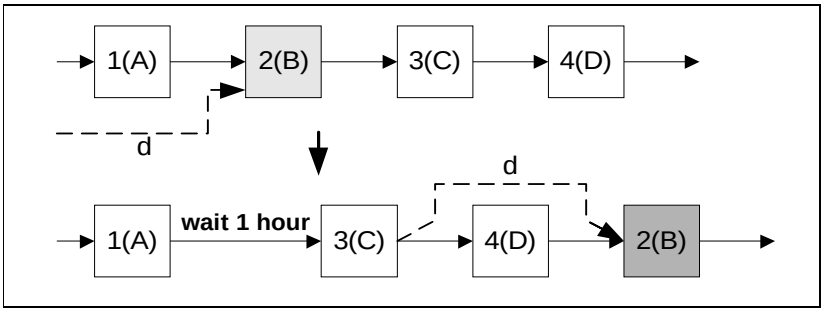


Abb. 5-11: Verzögerung eines Knotens durch Umsortieren.

Ist eine Verzögerung durch Umsortieren nicht möglich, muss eine Split/Join-Region mit AND-Split- und ALL-Join-Knoten in folgenden Schritten konstruiert werden:

- Einfügen des AND-Split- und des ALL-Join-Knotens,
- Umsetzen der vorhandenen und Einfügen neuer Kontrollfluss-Transitionen,
- Zuordnen der Verzögerungsbedingung zur eingehenden Kante des zu verzögernden Knotens.

Als erstes werden ein AND-Split-Knoten vor dem zu verzögernden Knoten und ein ALL-Join-Knoten nach dessen i-tem Nachfolger in die Workflow-Instanz eingefügt. Dabei werden die bestehenden Transitionen, in die die Knoten eingefügt werden, zu den ausgehenden Transitionen der neuen Knoten. Der Wert von i ist so zu wählen, dass die beiden Pfade der Split/Join-Region annähernd die gleiche Ausführungsdauer haben. Der zweite Pfad umfasst also so viele Knoten, wie in der Zeit *Ausführungsdauer des zu verzögernden Knoten + t* ausgeführt werden können.
Ist unter diesen wieder ein Knoten, der von der *delay*-Kontroll-Aktion betroffen ist, so endet der zweite Pfad vor diesem Knoten, um rekursive Abhängigkeiten zwischen den Umbauoperationen für die einzelnen Knoten zu vermeiden. Soll ein ebenfalls betroffener Knoten in den Parallelpfad aufgenommen werden, müsste zuerst dieser Knoten verzögert werden (über Umsortierung oder Split/Join-Region), damit die Knoten für den Parallelpfad des ersten zu verzögernden Knotens korrekt bestimmt werden können. Beispielsweise könnte es sein, dass der zweite zu verzögernde Knoten nach der Adaptation gar nicht mehr im Parallelpfad des ersten zu verzögernden Knotens liegt. Da für jeden betroffenen Knoten, der über eine Split/Join-Region zu verzögern ist, diese Problematik wieder auftreten kann, kann die Anzahl der Rekursionsstufen maximal so groß sein wie die Anzahl der betroffenen Knoten in der Adaptations-Region, wenn alle Knoten über eine Split/Join-Region zu verzögern sind und immer wieder ein betroffener Knoten im Parallelpfad des vorhergehenden betroffenen Knotens liegen würde. Eine Lösung für dieses Problem wäre eventuell, die betroffenen Knoten rückwärts zu sortieren, also den Knoten zuerst zu betrachten, der am nächsten zum Ende der Adaptations-Region liegt. Allerdings müsste bei diesem Ansatz erst überprüft werden, ob er eine Auswirkung auf die Korrektheit der übrigen Adaptationen hat, weshalb er in der vorliegenden Arbeit nicht weiter verfolgt wird.

Der AND-Split-Knoten wurde gewählt, damit auch weiterhin alle Knoten (der zu verzögernde und seine Nachfolger) ausgeführt werden. Mit dem ALL-Join-Knoten wird sichergestellt, dass erst dann der nächste Knoten im Kontrollfluss aktiviert wird, wenn alle Knoten der Split/Join-Region abgearbeitet wurden, wie es der Semantik der ursprünglich vorhandenen Sequenz entspricht.

In Abbildung 5-12 ist oben ein Workflow-Ausschnitt dargestellt, in dem der Aktivitätstyp B um *t* verzögert werden soll. Angenommen der betroffene Knoten 2 kann nicht verschoben werden und die Ausführung der Aktivität C dauert in etwa so lange wie die Ausführung des zu verzögernden Knotens (einschließlich der Verzögerung). Also ist eine Split/Join-Region zu konstruieren, deren Parallelpfad den Knoten 3 enthält. Im unteren Teil ist der Workflow-Ausschnitt nach dem Einfügen von Split- und Join-Knoten abgebildet. Die beiden Transitionen, in die die neuen Knoten eingefügt wurden, sind fett gezeichnet.

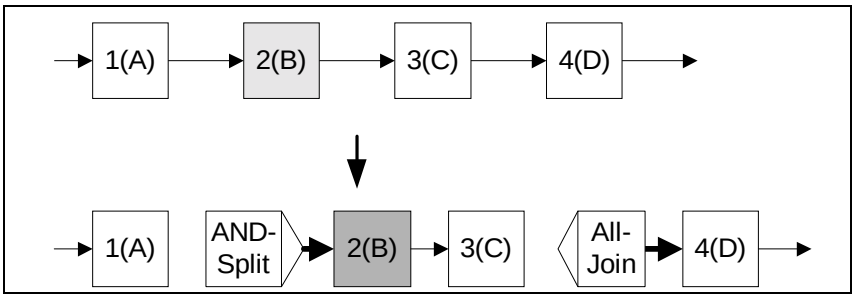


Abb. 5-12: Einfügen von Split- und Join-Knoten bei der Bearbeitung einer *delay*-Kontroll-Aktion.

Im nächsten Schritt werden die vorhandenen Kontrollfluss-Transitionen so umgesetzt und durch neue ergänzt, dass eine korrekte Split/Join-Region entsteht. Die bestehende Kante zwischen dem zu verzögernden Knoten und seinem Nachfolger wird so umgesetzt, dass sie vom zu verzögernden Knoten zum Join-Knoten führt. Anschließend werden zwischen folgenden Knoten neue Transitionen eingefügt:

- zwischen dem ursprünglichen Vorgänger des zu verzögernden Knotens und dem Split-Knoten,
- zwischen dem Split-Knoten und dem ersten Knoten des Parallelpfades,

- zwischen dem letzten Knoten im Parallelpfad und dem Join-Knoten.

Abschließend muss der eingehenden Kontrollfluss-Transition des zu verzögernden Knotens eine neue temporale Transitions-Bedingung zugeordnet werden, die die eigentliche Verzögerung des Knotens bewirkt. Dabei ist eine eventuell schon vorhandene temporale Transitions-Bedingung (etwa *wait 1 hour*) zu berücksichtigen. Die neue Transitions-Bedingung lautet dann entweder *wait t* oder *wait (t + x)*, wobei *t* die Verzögerung und *x* eine möglicherweise schon vorhanden Wartezeit (z. B. 1 hour) bezeichnet.
Abbildung 5-13 zeigt den Workflow-Ausschnitt aus Abbildung 5-12 nach der Durchführung der letzten beiden Schritte (Einfügen von Kontrollfluss-Transitionen und einer Transitions-Bedingung). Die neu eingefügten Transitionen sind fett, die umgesetzten Transitionen gestrichelt gezeichnet.

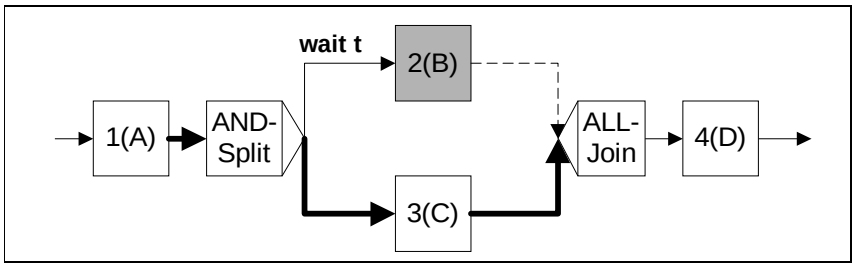


Abb. 5-13: Verzögerung eines Knotens durch eine Split/Join-Region.

Durch das Herausziehen des zu verzögernden Knotens aus der Sequenz in einen Parallelpfad, werden die Knoten, die hinter der neu konstruierten Split/Join-Region liegen, möglicherweise früher ausgeführt. Ist unter diesen kein von einer *delay*-Kontroll-Aktion betroffener Knoten, muss dieser Fall nicht weiter berücksichtigt werden. Liegen allerdings weitere zu verzögernde Knoten hinter der Split/Join-Region, so muss die Zeit, um die sie früher ausgeführt werden würden, zu ihrer Verzögerung *t* addiert werden. Damit werden diese Knoten ausgehend von ihrem ursprünglichen Ausführungszeitpunkt korrekt verzögert.
Bei der Bestimmung der i Knoten für den Parallelpfad muss auch berücksichtigt werden, ob es Regeln gibt, die die sequentielle Ausführung der zu verzögernden Aktivität mit Aktivitäten aus dem potentiellen Parallelpfad vorschreibt. Auch eine ausgehende interne Objektfluss-Transition vom zu verzögernden Knoten zu einem seiner i Nachfolger erzwingt eine sequentielle Aus-

führung dieser beiden Knoten. In diesem Fall sollte die Split/Join-Region vor solchen Zielknoten von Objektfluss- oder potentiellen Synchronisations-Transitionen enden, auch wenn die Ausführungsdauer des Parallelpfades dann kürzer ist als die des Pfades mit dem verzögerten Knoten. Durch die sequentielle Ausführung wird der Zielknoten einer Synchronisations- oder Objektfluss-Transition sowieso erst gestartet, wenn die Ausführung des zu verzögernden Knotens beendet ist.[1]

5.1.6 Kontroll-Aktion *add*(A)

Eine *add*-Kontroll-Aktion besagt, dass innerhalb ihres Gültigkeitsintervalls einmal eine Aktivität vom Typ A einmal zusätzlich zu den schon in der Workflow-Instanz enthaltenen Aktivitäten ausgeführt werden soll. Es muss also in der Adaptations-Region eine passende Stelle gefunden werden, an der ein Aktivitätsknoten eingefügt werden kann, dem dann eine Aktivität vom Typ A zugeordnet wird.

Da das Einfügen eines neuen Knotens die Knotenmenge der Adaptations-Region beeinflussen kann (vgl. Abschnitt 5.2), muss versucht werden, diese Beeinflussung möglichst gering zu halten. Der neue Knoten darf deshalb nicht in eine bestehende Sequenz eingefügt werden, wenn dadurch ein von einer gültigen Kontroll-Aktion betroffener Knoten erst nach Ende des Gültigkeitsintervall *T* ausgeführt werden würde.

Wäre das der Fall, wird bei einer *add*-Kontroll-Aktion der neue Knoten über eine Split/Join-Region mit zwei Pfaden und AND-Split- bzw. ALL-Join-Knoten konstruiert. Ein Pfad der Region enthält den neu einzufügenden Knoten, der andere einen oder mehrere Aktivitäts- oder Kommunikationsknoten der Adaptations-Region. Die Split/Join-Region darf nicht in eine OR-Verzweigung eingefügt werden, deren Verzweigungsbedingungen noch nicht ausgewertet werden können, da dann nicht gewährleistet ist, dass der Pfad, in dem die neue Split/Join-Region liegt, wirklich ausgeführt wird. Außerdem muss es erlaubt sein, die Aktivitäten des neuen Knotens und der schon vorhandenen Knoten im Parallelpfad parallel auszuführen.

1. Die Bearbeitung einer *delay*-Kontroll-Aktion ist mit einem rein prozeduralen Ansatz kaum vollständig zu bewältigen, da häufig Abhängigkeiten in Form von Regeln oder Objektfluss-Transitionen zwischen den einzelnen Aktivitätstypen berücksichtigt werden müssen. Eventuell wäre constraint-Programming sinnvoll einsetzbar.

Zur Bearbeitung einer *add*-Kontroll-Aktion ist zuerst eine passende Einfügestelle zu finden. Kann der Knoten in keine der bestehenden Sequenzen eingefügt werden, weil dadurch von Kontroll-Aktionen betroffene Knoten erst nach Ende des Gültigkeitsintervalls zur Ausführung kämen oder weil es Regeln gibt, die die Ausführung der Aktivität A in den vorhandenen Sequenzen verbieten, muss eine Stelle zum Einfügen der Split/Join-Region gefunden werden.
Dazu werden die einzelnen Pfade der Adaptations-Region solange durchsucht, bis ein oder mehrere Knoten für den Parallelpfad gefunden wurden. Verläuft diese Suche ergebnislos, weil beispielsweise die Adaptations-Region sehr klein ist, kann der Parallelpfad nicht automatisch bestimmt werden. Um den neuen Knoten dennoch einfügen zu können, muss eine Nutzeranfrage generiert werden, bei der dem Nutzer die Adaptations-Region angezeigt wird und er einen oder mehrere Knoten auswählen kann, die den Parallel-Pfad bilden sollen. Dadurch wird sichergestellt, dass der neue Knoten auf jeden Fall eingefügt werden kann. Für den Parallelpfad werden nach Möglichkeit so viele Knoten ausgewählt, dass die beiden Pfade der neuen Split/Join-Region ungefähr die gleiche Ausführungsdauer haben.

Wird der neue Knoten in eine bestehende Sequenz eingefügt, sind folgende Schritte durchzuführen:

- Einfügen des neuen Knotens in die bestehende Transition,
- Einfügen einer neuen Kontrollfluss-Transition,
- Einfügen von Transitions-Bedingungen,
- Einfügen eventuell benötigter Synchronisations- und Objektfluss-Transitionen.

Der neue Knoten wird zuerst in die ausgewählte Transition eingefügt, dabei wird diese zur ausgehenden Transition des neuen Knotens. Danach wird eine neue Kontrollfluss-Transition als eingehende Kante des neuen Knotens eingefügt. Gibt es Bedingungen für den Übergang vom Vorgängerknoten zum neuen Knoten bzw. vom neuen Knoten zum Nachfolgerknoten, so sind diese über Transitions-Bedingungen für die ein- bzw. ausgehende Kontrollfluss-Transition des neuen Knotens zu modellieren.
Als Nächstes muss überprüft werden, ob die neue Aktivität mit anderen Aktivitäten in möglichen Parallelpfaden synchronisiert werden muss. Ist das der Fall, werden Synchronisations-Transitionen eingefügt. Benötigt die neue Aktivität Eingabedaten, so müssen diese über eine

interne (wenn passender Quellknoten vorhanden) oder externe Datenfluss-Transition zur Verfügung gestellt werden. Produziert sie auch Daten, so wird zusätzlich noch eine externe ausgehende Datenfluss-Transition eingefügt.

In Abbildung 5-14 ist im oberen Workflow-Ausschnitt eine Sequenz dargestellt, in die die Aktivität A eingefügt werden soll. Die passende Transition ist die zwischen den Knoten 2 und 3. Der mittlere Workflow-Ausschnitt zeigt die Sequenz nach dem Einfügen des neuen Knotens, der neuen Kontrollfluss-Transition und einer Transitions-Bedingung, die besagt, dass zwischen der Aktivität C und der Aktivität A immer eine Stunde gewartet werden muss.

Im untersten Abschnitt wurden dann noch Objektfluss-Transitionen eingefügt. Die Aktivität A benötigt die Daten D1, die die Aktivität B zur Verfügung stellt. Es muss also eine interne Datenfluss-Transition eingefügt werden. Außerdem produziert die neue Aktivität die Daten D2, die über eine externe Datenfluss-Transition in eine Datenbank ausgeschrieben werden, damit sie nicht verloren gehen.

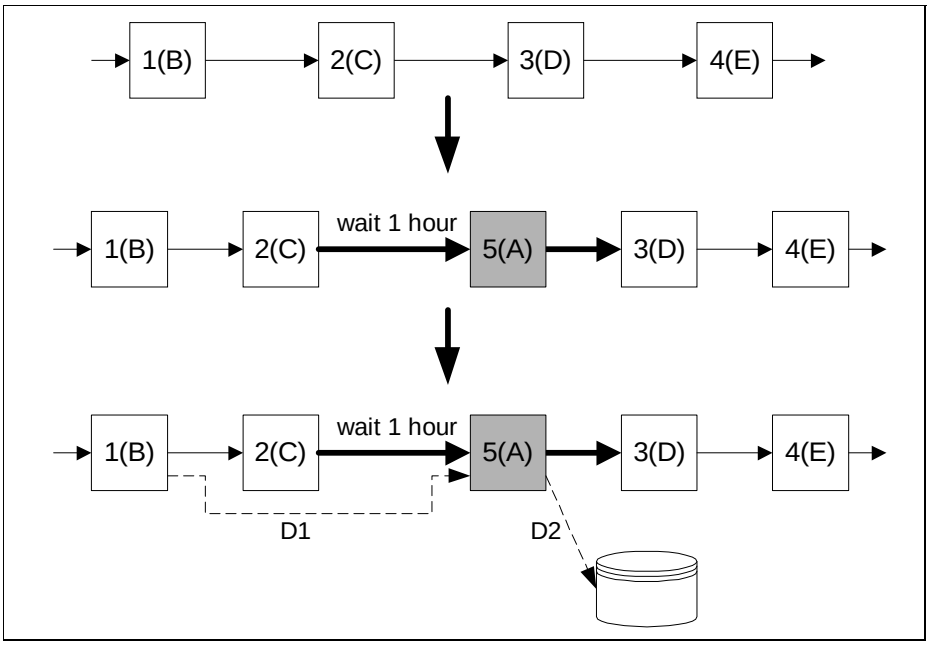


Abb. 5-14: Einfügen eines neuen Knotens in eine Sequenz.

Muss der neue Knoten über eine Split/Join-Region eingefügt werden, sind folgende Schritte nötig:

- Konstruktion der Split/Join-Region durch Einfügen von AND-Split-, ALL-Join- und neuem Aktivitätsknoten und neuen Kontrollfluss-Transitionen,
- Einfügen eventuell nötiger Synchronisations-Transitionen,
- Einfügen möglicherweise benötigter Objektfluss-Transitionen.

Als Erstes wird um den oder die Knoten des Parallelpfades die Split/Join-Region mit dem neuen Aktivitätsknoten konstruiert. Zur Verdeutlichung dieses Schrittes ist in Abbildung 5-15 ganz oben ein Workflow-Ausschnitt abgebildet, in den ein neuer Aktivitätsknoten mit dem Aktivitätstyp A eingefügt werden soll. Als einziger Knoten des Parallel-Pfades wurde Knoten 2 ausgewählt. Neu entstandene oder neu eingefügte Transitionen sind fett gezeichnet.
Zuerst werden der AND-Split-Knoten in die eingehende Kontrollfluss-Transition des ersten Knotens des Parallelpfades und der ALL-Join-Knoten in die ausgehende Kontrollfluss-Transition des letzten Knotens des Parallel-Pfades eingefügt. Die bestehenden Transitionen werden dabei jeweils zur ausgehenden Transition der neuen Knoten. Zusätzlich wird eine neue Transition zwischen dem Split- und dem Join-Knoten gezogen, in die dann der neue Aktivitätsknoten eingefügt wird (vgl. Abbildung 5-15 den zweiten und dritten Workflow-Ausschnitt). Abschließend werden dann noch folgende Kontrollfluss-Transitionen neu eingefügt:

- die eingehende Transition des Split-Knotens,
- die ausgehende Transition des letzten Knotens des Parallel-Pfades und
- die eingehende Transition des neuen Aktivitätsknotens.

Das Ergebnis ist eine korrekte Split/Join-Region, wie sie in Abbildung 5-15 im untersten Workflow-Ausschnitt dargestellt ist.

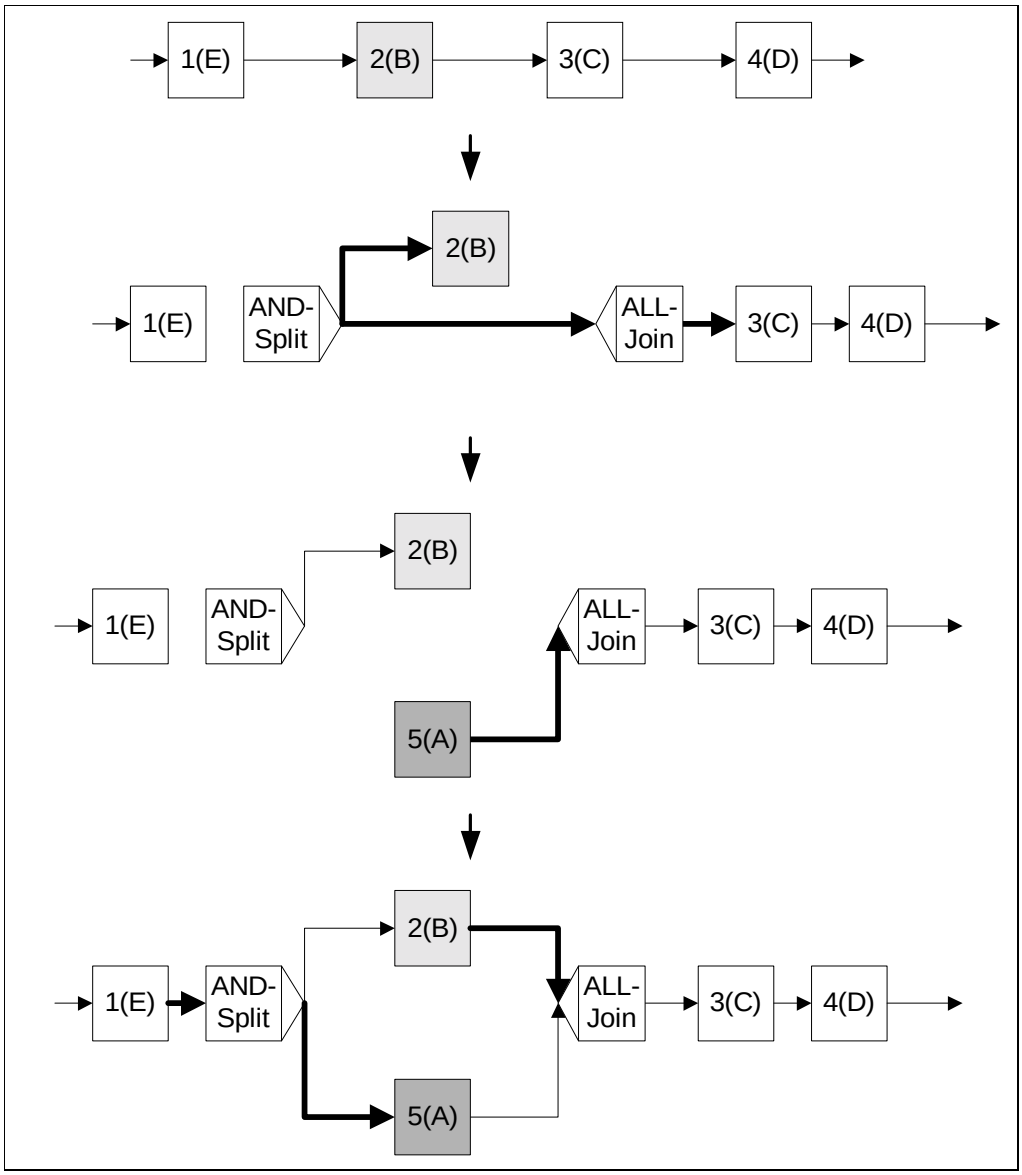


Abb. 5-15: Konstruktion einer Split/Join-Region zum Einfügen eines neuen Knotens.

Liegt die neu eingefügte Split/Join-Region in einem Pfad einer anderen Split/Join-Region muss der neu eingefügte Aktivitätsknoten eventuell mit anderen Aktivitätsknoten synchronisiert werden. Dazu müssen zunächst über die Wissensbasis alle Aktivitätstypen bestimmt werden, mit denen der neue Aktivitätstyp A zu synchronisieren ist. Danach sind alle zu dem Einfügepfad parallelen Pfade zu untersuchen, ob darin noch nicht ausgeführte Aktivitätsknoten mit den entsprechenden Aktivitätstypen vorkommen. Wenn ja, müssen die passenden ein- und ausgehenden Synchronisations-Transitionen eingefügt werden.

Im letzten Schritt muss überprüft werden, ob die neu eingefügte Aktivität Daten benötigt oder produziert. Benötigt die neue Aktivität Daten, muss ein im Kontrollfluss vor dem neuen Knoten liegender und noch nicht aktivierter Knoten gefunden werden. Von diesem wird dann eine interne Datenfluss-Transition zu dem neuen Knoten gezogen. Gibt es keinen solchen Knoten, muss eine externe Datenfluss-Transition eingefügt werden, über die die entsprechenden Daten aus einer Datenbank eingelesen werden. Wenn die Aktivität Daten erzeugt, werden diese über eine externe Datenfluss-Transition in eine passende Datenbank ausgeschrieben.
Abbildung 5-16 zeigt nochmals das obige Beispiel mit zusätzlich eingefügten Datenfluss-Transitionen. Die neue eingefügte Aktivität A benötigt die Daten D1, die von der Aktivität E zur Verfügung gestellt werden. Es ist also eine interne Datenfluss-Transition zwischen Knoten 1 und Knoten 5 zu ziehen. Außerdem produziert die neue Aktivität die Daten D2, die über eine externe Datenfluss-Transition in eine Datenbank ausgeschrieben werden.

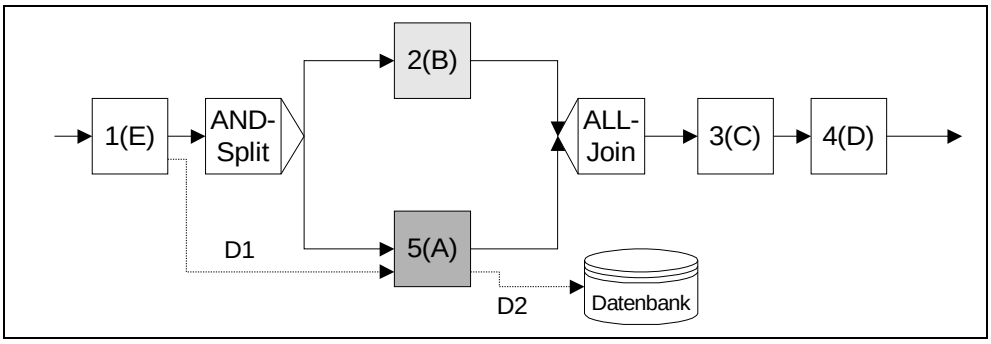


Abb. 5-16: Einfügen von Objektfluss-Transitionen bei der Bearbeitung einer *add*-Kontroll-Aktion.

5.2 Verschiebung der Adaptations-Region

Durch die Änderungen an den Knoten der Adaptations-Region AR_T kann sich die ursprüngliche Knotenmenge der Adaptations-Region nach Abschluss der Adaptationen verkleinert oder vergrößert haben. Die Knotenmenge von AR_T hat sich verkleinert, wenn Knoten (z. B. durch eine *drop*-Kontroll-Aktion) gelöscht wurden, also weniger Knoten als vorher innerhalb *T* ausgeführt werden sollen. Die Knotenmenge hat sich vergrößert, wenn zusätzliche Knoten (bei-

spielsweise durch eine *add*-Kontroll-Aktion) eingefügt wurden, also mehr Knoten innerhalb *T* ausgeführt werden sollen.

Im ersten Fall (Verkleinerung der Knotenmenge von AR_T) wird die Ausführung der in AR_T verbleibenden Knoten im Allgemeinen nicht das ganze Gültigkeitsintervall *T* in Anspruch nehmen. Es können also möglicherweise Knoten, die bisher außerhalb der Adaptations-Region lagen und nicht von einer gültigen Kontroll-Aktion betroffen sind, früher zur Ausführung kommen. In dem Workflow-Ausschnitt aus Abbildung 5-17 könnte z. B. Knoten 4 früher ausgeführt werden, da er nicht von der gültigen Kontroll-Aktion betroffen ist.

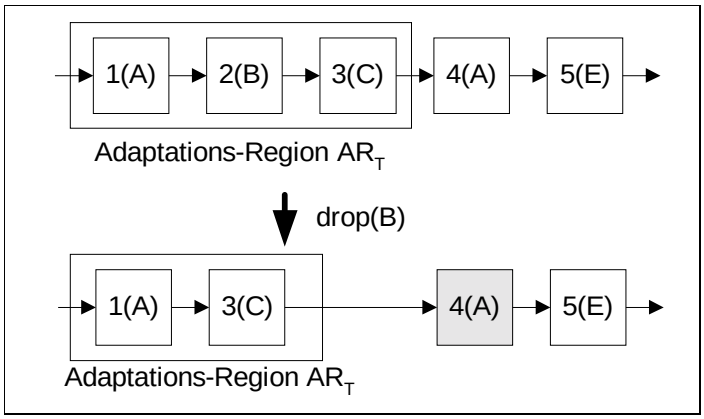


Abb. 5-17: Verkleinerung der Knotenmenge der Adaptations-Region durch eine *drop*-Kontroll-Aktion.

Es dürfen aber keine Knoten früher ausgeführt werden, die von einer der gültigen Kontroll-Aktionen betroffen sind und außerhalb der ursprünglichen Knotenmenge der Adaptations-Region AR_T liegen. Würden sie früher ausgeführt als geplant, müssten sie ebenfalls adaptiert werden, was zu unerwünschten Effekten führen kann.

Angenommen ein Workflow enthält eine Sequenz, in der siebenmal nacheinander die Aktivität A ausgeführt wird (vgl. Abbildung 5-18). Für diesen Workflow wird die Kontroll-Aktion *drop(A)@T* gefolgert, wobei innerhalb *T* die ersten vier A-Aktivitäten ausgeführt werden können und deshalb gelöscht werden. Würde man jetzt die Knoten 5 bis 7 früher ausführen, würden sie innerhalb *T* zur Ausführung kommen und müssten ebenfalls gelöscht werden. Es würden also alle sieben A-Aktivitäten gelöscht und nicht nur die ersten vier, wie es der Semantik der Kontroll-Aktion entspräche.

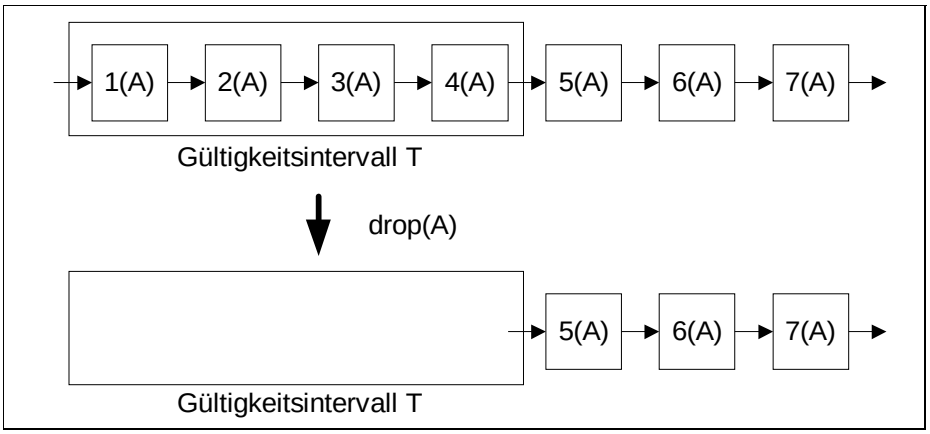


Abb. 5-18: Korrekte Durchführung der Kontroll-Aktion *drop(A)* in einer Sequenz von gleichen Aktivitäten.

Sind der erste Knoten außerhalb der Adaptations-Region von einer gültigen Kontroll-Aktion betroffen, seine Nachfolger aber nicht, so können diese eventuell früher ausgeführt werden. Dazu muss die ursprüngliche Reihenfolge der Knoten geändert werden. Das ist nur dann möglich, wenn keine Regeln verletzt werden, die z. B. eine feste Reihenfolge bestimmter Aktivitäten vorschreiben. Außerdem muss sichergestellt werden, dass der Datenfluss weiterhin korrekt bleibt. Wenn im in Abbildung 5-19 dargestellten Beispiel Knoten 7 Daten von Knoten 5 bekommen würde, kann er nicht vor diesem ausgeführt werden, auch wenn die Zeit für seine frühere Ausführung vorhanden wäre.

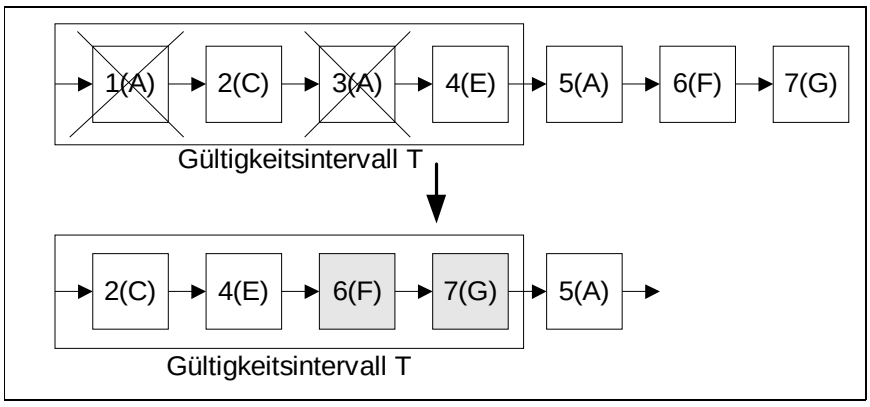


Abb. 5-19: Frühere Ausführung von Knoten, die nicht von der Kontroll-Aktion *drop(A)* betroffen sind und deren Umordnung keine Regeln verletzt und den Datenfluss korrekt erhält.

Hat sich die Knotenmenge der Adaptations-Region z. B. durch eine *add*-Kontroll-Aktion vergrößert, kommt ein Teil der Knoten möglicherweise nicht mehr innerhalb des Gültigkeitsintervalls *T* zur Ausführung. Sind diese Knoten nicht von einer der gültigen Kontroll-Aktionen betroffen, ist es unerheblich, ob sie innerhalb *T* oder erst danach ausgeführt werden. So wird im Beispiel aus Abbildung 5-20 Knoten 3 durch das Einfügen des neuen Knotens 6 voraussichtlich nicht mehr innerhalb *T* ausgeführt werden. Gilt für diesen Knoten keine Kontroll-Aktion, sind keine weiteren Maßnahmen zu treffen. Wurde der Knoten aber aufgrund einer anderen Kontroll-Aktion bereits adaptiert und kommt nicht mehr innerhalb *T* zur Ausführung, muss die Adaptation möglicherweise rückgängig gemacht werden. Da dies im Allgemeinen nicht automatisch entschieden werden kann, werden solche Knoten bei der abschließenden Überarbeitung der Änderungen für den Nutzer gekennzeichnet, so dass dieser entscheiden kann, ob die Änderung korrekt ist oder zurückgenommen werden muss.

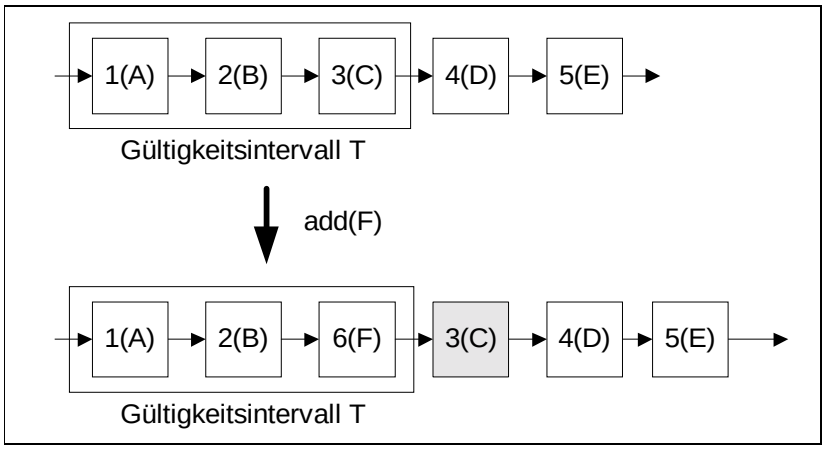


Abb. 5-20: Ausführung von Knoten außerhalb des Gültigkeitsintervalls *T* durch die Bearbeitung einer *add*-Kontroll-Aktion.

Ein weiterer Fall, der berücksichtigt werden muss, tritt dann auf, wenn durch die Vergrößerung der Knotenmenge Knoten nicht mehr innerhalb von *T* zur Ausführung kommen, die von einer noch nicht bearbeiteten Kontroll-Aktion betroffen sind. Angenommen im obigen Beispiel (Abb. 5-20) wäre Knoten 3 von der Kontroll-Aktion *drop(C)* betroffen, müsste also eigentlich gelöscht werden. Durch die Bearbeitung der *add*-Kontroll-Aktion wird er aber nicht mehr innerhalb *T* ausgeführt, ist also eigentlich bereits gelöscht (er wird im Intervall *T* nicht ausge-

führt). Dieses *Löschen durch Verschieben* kann problematisch werden, wenn die Aktivität C z. B. die Verabreichung eines Medikaments darstellt. Die Semantik der Kontroll-Aktion *drop(C)* fordert, dass das Medikament überhaupt nicht gegeben werden soll, und nicht, dass die Gabe verzögert werden soll. Wird der betroffene Knoten nur verschoben und nicht gelöscht, erhält der Patient eine Dosis zuviel, was unter Umständen negative Auswirkungen haben kann. Soll eine Aktivität nur verzögert nicht gelöscht werden, muss eine *delay*-Kontroll-Aktion gefolgert werden.

Die obige Diskussion zeigt, dass durch die Adaptation der Workflow-Instanz Aktivitäten eher oder später zur Ausführung kommen können als geplant. Gelten mehrere Kontroll-Aktionen gleichzeitig, können sich die Auswirkungen der Adaptationen gegenseitig aufheben, so dass am Ende doch alle Knoten zu den geplanten Zeiten ausgeführt werden. Im Folgenden wird für jede Kontroll-Aktion einzeln gezeigt, welchen Einfluss sie auf die Knotenmenge der Adaptations-Region AR_T hat. Danach wird erläutert, in welcher Reihenfolge die Adaptationen bei mehreren gleichzeitig gültigen Kontroll-Aktionen vorgenommen werden.

- *check(A)@T*:
  Die *check*-Kontroll-Aktion hat keinen Einfluss auf die Knoten der Adaptations-Region, da noch keine Änderungen vorgenommen werden. Der Nutzer kann nur angeben, welche Kontroll-Aktion auf einen betroffenen Knoten angewendet werden soll.

- *replace(A, B)@T mit Dauer(B) >= Dauer(A)*:
  Wenn bei einer *replace*-Kontroll-Aktion die neue Aktivität länger dauert als die alte, können Knoten, die vor Anwendung der Kontroll-Aktion in der Adaptations-Region lagen, danach möglicherweise nicht mehr innerhalb des Gültigkeitsintervalls *T* ausgeführt werden (vgl. Abbildung 5-21).

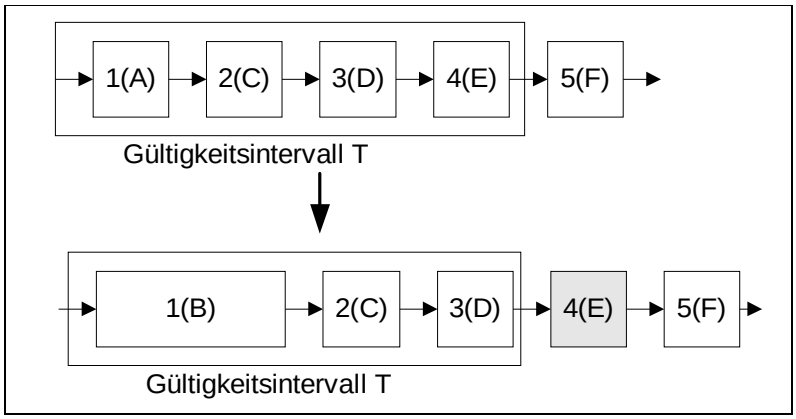


Abb. 5-21: Verdrängung von Knoten aus der Adaptations-Region nach *replace(A, B)@T* mit Dauer(B) >= Dauer(A).

- *replace(A, B)@T mit Dauer(B) < Dauer(A)*:
  Ist die neue Aktivität kürzer als die alte, können eventuell Knoten, die außerhalb der Adaptations-Region liegen, früher ausgeführt werden, falls sie nicht von einer Kontroll-Aktion betroffen sind, durch eine mögliche Umsortierung (vgl. obige Diskussion zur Verkleinerung der Knotenmenge der Adaptations-Region) keine Regeln bzgl. der Reihenfolge von Aktivitäten verletzt werden und der Datenfluss korrekt bleibt (vgl. Abbildung 5-22).

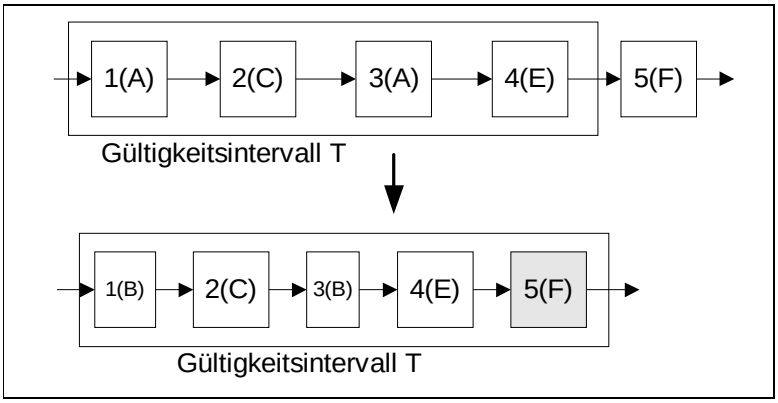


Abb. 5-22: Frühere Ausführung von Knoten außerhalb der Adaptations-Region nach *replace(A, B)@T* mit Dauer(B) < Dauer(A).

- *add(A)@T*:

  Zur Umsetzung einer *add*-Kontroll-Aktion wird der neue Knoten entweder an einer passenden Stelle in eine vorhandene Sequenz oder in einen neuen Parallelpfad zur Adaptations-Region AR_T eingefügt (vgl. Abschnitt 5.1.6).

  Wird der neue Knoten in eine vorhandene Sequenz eingefügt, kann sich die Knotenmenge der Adaptations-Region AR_T vergrößern, d. h. einige Knoten werden möglicherweise nicht mehr innerhalb *T* ausgeführt (vgl. Abb. 5-23 den unteren Workflow-Ausschnitt). Wird ein Parallelpfad konstruiert, werden im Allgemeinen (bei günstiger Wahl der Knoten des zweiten Pfades der Split-Join-Region) keine Knoten aus AR_T herausgedrückt (vgl. Abb. 5-23 den mittleren Workflow-Ausschnitt).

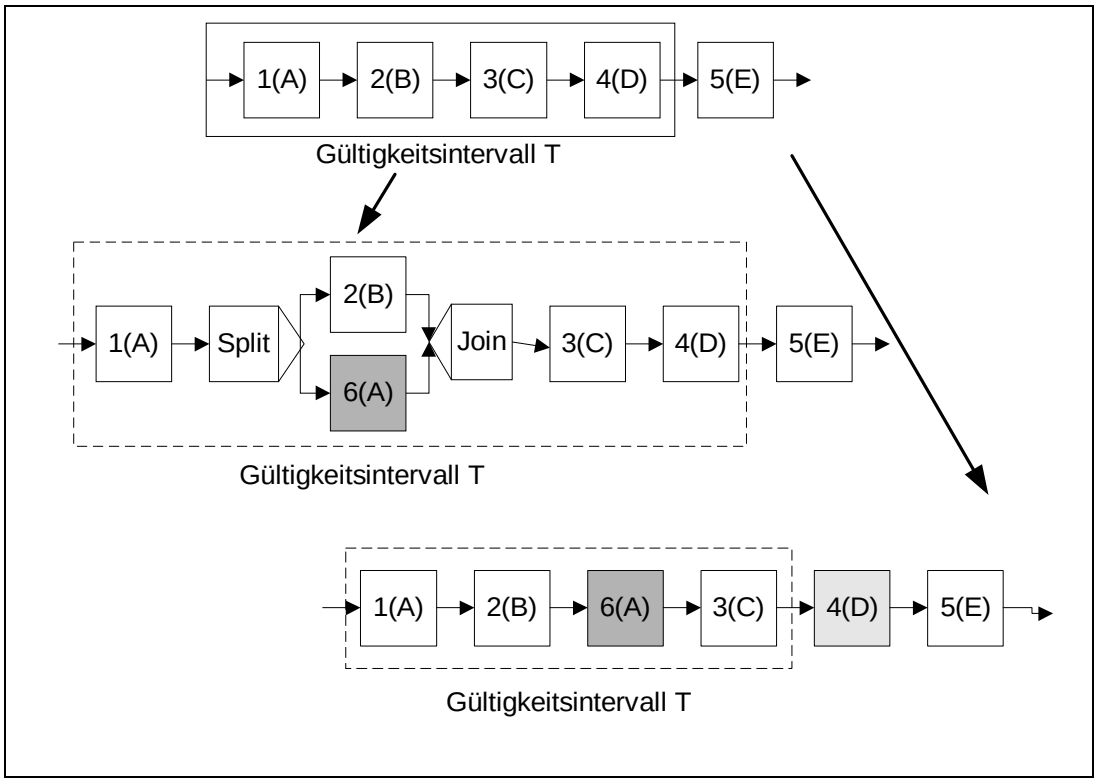


Abb. 5-23: Veränderung der Adaptations-Region bei *add(A)@T.*

- *drop(A)@T*:

  Durch das Löschen von Knoten verkleinert sich die Knotenmenge der Adaptations-Region AR_T auf jeden Fall. Ob allerdings Knoten außerhalb von AR_T früher ausgeführt werden können, hängt davon ab, ob die gelöschten Knoten Einfluss auf die gesamte Ausführungsdauer von AR_T haben.

  Das ist dann der Fall, wenn sie in einer Sequenz (vgl. Abb. 5-24) oder in einem Splitpfad liegen, der die Dauer der gesamten Split/Join-Region bestimmt. Bei einer Split/Join-Region mit ALL-Join-Knoten ist das der längste, bei einer Split/Join-Region mit One-Join-Knoten der kürzeste Pfad. Wird durch das Löschen von Knoten die Ausführungsdauer des bestimmenden Pfades kürzer, verkürzt sich die Dauer der gesamten Split/Join-Region und u. U. auch die von AR_T. Bei einer Split/Join-Region mit One-Join-Knoten kann durch das Löschen von Knoten auch ein anderer Pfad der kürzeste Pfad werden. Dann verkürzt sich die Dauer der gesamten Split/Join-Region und damit u. U. die von AR_T ebenfalls. In beiden Fällen können eventuell Knoten früher ausgeführt werden.

  Wenn Knoten aus solchen Pfaden gelöscht werden, die die Ausführungsdauer der gesamten Adaptations-Region nicht beeinflussen, können keine Knoten außerhalb der Adaptations-Region früher ausgeführt werden.

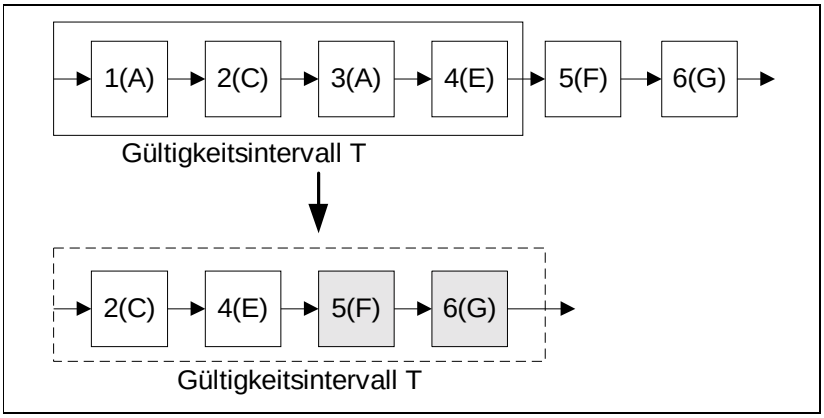


Abb. 5-24: Frühere Ausführung von Knoten außerhalb der Adaptations-Region nach *drop(A)@T*.

- *delay(A, t)@T*:

  Die Umsetzung einer *delay*-Kontroll-Aktion kann auf zwei Arten erfolgen: Entweder durch Umsortierung oder durch Konstruktion einer Split/Join-Region, deren einer Pfad den zu verzögernden Knoten enthält (vgl. Abschnitt 5.1.5).

  Wird der Knoten durch eine Umsortierung der Knoten der Adaptations-Region AR_T verzögert, ändert sich die Knotenmenge nicht (vgl. Abb. 5-25 den mittleren Workflow-Ausschnitt).

  Wird der Knoten in einen Parallelpfad herausgezogen (vgl. Abb. 5-25 unterer Workflow-Ausschnitt), können möglicherweise Knoten außerhalb der Adaptations-Region früher zur Ausführung kommen.

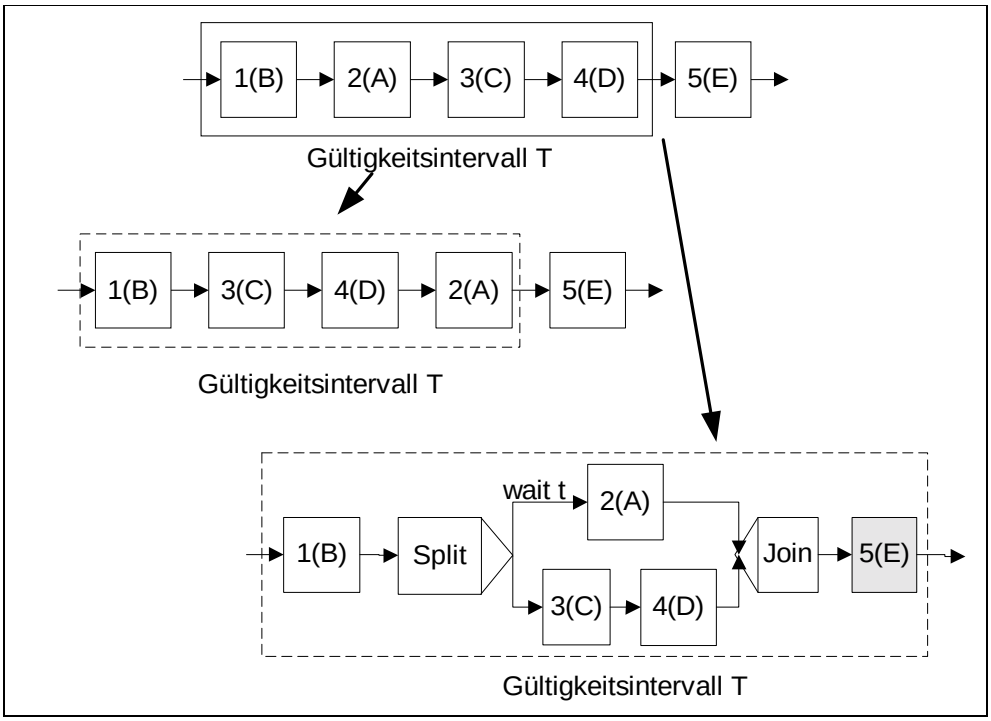


Abb. 5-25: Veränderung der Adaptations-Region bei *delay(A, t)@T.*

Da sich die Auswirkungen der Adaptationen gegenseitig aufheben können, wird bei der Bearbeitung mehrerer gleichzeitig gültiger Kontroll-Aktionen zugelassen, dass die Knotenmenge der Adaptations-Region während der Bearbeitung schwankt. Dazu werden alle gültigen Kontroll-Aktionen ausgehend von der ursprünglichen Adaptations-Region bearbeitet und erst

danach wird überprüft, ob Knoten früher ausgeführt werden können oder ob Knoten aus der Adaptations-Region herausgeschoben wurden. Würde man diese Überprüfung nach jeder Kontroll-Aktion durchführen, würde u. U. Rechenzeit verschwendet werden. Wenn z. B. zuerst ein Knoten durch eine *drop*-Kontroll-Aktion gelöscht wird, kann man einen oder mehrere andere Knoten früher ausführen. Wird danach eine *add*-Kontroll-Aktion bearbeitet, liegen möglicherweise genau diese Knoten doch wieder außerhalb der Adaptations-Region, da ein Knoten neu eingefügt wurde. Also waren die Überprüfungen nach der ersten Kontroll-Aktion überflüssig.
Die gleichzeitig gültigen Kontroll-Aktionen können in einer beliebigen Reihenfolge bearbeitet werden. Um Entscheidungen, z. B. über die passende Einfügestelle für einen Knoten (bei *add*- oder *delay*-Kontroll-Aktionen), nicht korrigieren zu müssen, ist es aber sinnvoll, die Kontroll-Aktionen *check, drop* und *replace* zuerst durchzuführen, da diese den Kontrollfluss so verändern können, dass die Einfügestelle nicht mehr passend ist.[1]

Wie dieser Abschnitt verdeutlicht, sind die Abhängigkeiten zwischen gleichzeitig gültigen Kontroll-Aktionen sehr komplex. Dadurch wird ihre automatische Behandlung erschwert und muss an den passenden Stellen durch eine Interaktion mit dem Nutzer ergänzt werden. Eine weitere Diskussion zu den Abhängigkeiten zwischen Kontroll-Aktionen findet sich auch in [Mue00].

5.3 Zurücksetzen von Adaptationen

Kommen bei der weiteren Ausführung der Workflow-Instanz nicht alle Knoten aus der geschätzten Adaptations-Region innerhalb des Gültigkeitsintervalls zur Ausführung, kann es erforderlich sein, Adaptationen rückgängig zu machen.
Abbildung 5-26 verdeutlicht die Situation. Angenommen in der geschätzten Adaptations-Region, die die Knoten 1 bis 8 umfasst, waren die Knoten 1, 3 und 7 von Kontroll-Aktionen betroffen und wurden adaptiert. Bei der weiteren Ausführung der Instanz zeigt sich aber, dass die Adaptations-Region zu groß geschätzt wurde und deshalb der geänderte Knoten 7 nicht

1. Die Aussagen des letzten Absatzes basieren auf dem aktuellen Stand des Projektes. Möglicherweise zeigt sich während der weiteren Entwicklung des Systems AGENTWORK, dass es vorteilhaft ist, bei der Bearbeitung gleichzeitig gültiger Kontroll-Aktionen eine bestimmte Reihenfolge einzuhalten.

mehr innerhalb des Gültigkeitsintervalls *T* der entsprechenden Kontroll-Aktion ausgeführt werden wird. Die Änderungen, die an diesem Knoten vorgenommen wurden, müssen also wieder rückgängig gemacht werden.

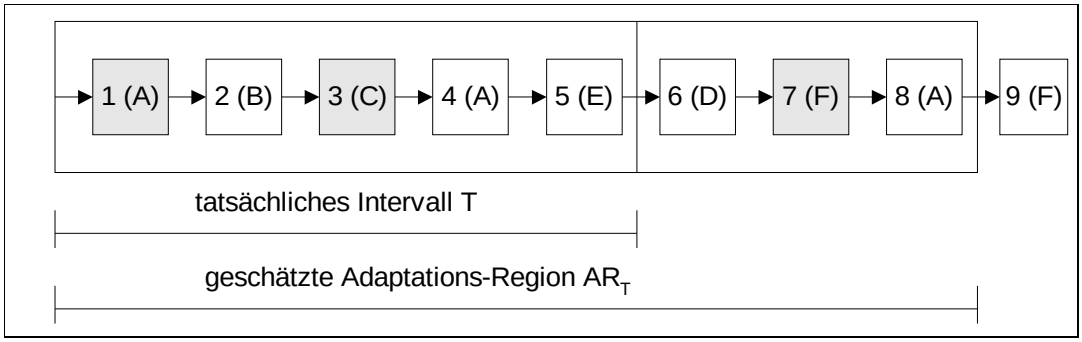


Abb. 5-26: Adaptations-Region zu groß geschätzt.

Wurde eine Adaptations-Region zu groß geschätzt, muss für jede Kontroll-Aktion, die für diese Adaptations-Region galt, überprüft werden, ob Knoten geändert wurden, die nicht mehr innerhalb des Gültigkeitsintervalls zur Ausführung kommen. Existieren solche Knoten, müssen für sie die Änderungen zurückgenommen werden.
Im Folgenden wird für die einzelnen Kontroll-Aktionen *drop*, *replace*, *delay* und *add* dargestellt, wie Änderungen rückgängig gemacht werden. *Check*-Kontroll-Aktionen können nicht rückgängig gemacht werden, da bei ihrer Bearbeitung keine Änderungen am Kontroll- oder Datenfluss vorgenommen werden (vgl Abschnitt 5.1.2).

- *drop-Kontroll-Aktion:*
  Galt für die zu groß geschätzte Adaptations-Region eine *drop*-Kontroll-Aktion, wurden möglicherweise Knoten gelöscht, die jetzt außerhalb des Gültigkeitsintervalls liegen würden. Sie müssen also wieder eingefügt werden.
  Um festzustellen, ob solche Knoten existieren, müssen die Knoten der Adaptations-Region bestimmt werden, die jetzt nicht mehr innerhalb des Gültigkeitsintervalls zur Ausführung kommen. Sind darunter solche, die von der Kontroll-Aktion betroffen waren und deshalb gelöscht wurden, müssen sie wieder in die Workflow-Instanz eingefügt werden.

Falls vor dem Löschen der Knoten der Datenfluss umgelenkt wurde (vgl. Abschnitt 5.1.3), müssen diese Änderungen falls möglich auch wieder rückgängig gemacht werden. Dabei können eingehende interne Datenfluss-Transitionen nur dann wieder eingefügt werden, wenn der Quellknoten der Transition noch im Zustand Untouched ist. Ist das nicht der Fall, müssen die benötigten Daten über eine externe Datenfluss-Transition aus der entsprechenden Datenbank eingelesen werden.

- *replace-Kontroll-Aktion:*
  Wurde für Knoten aus der Adaptations-Region, die nicht mehr innerhalb des Gültigkeitsintervalls zur Ausführung kommen, eine *replace*-Kontroll-Aktion durchgeführt, wird diese dadurch rückgängig gemacht, dass dem Knoten wieder der ursprüngliche Aktivitätstyp zugeordnet wird. Wurden Transitions-Bedingungen an den ein- oder ausgehenden Kontrollfluss-Transitionen geändert, müssen wieder die ursprünglichen Bedingungen eingesetzt werden. Falls Synchronisations- oder Objektfluss-Transitionen gelöscht oder neu gezogen wurden (vgl. Abschnitt 5.1.4), sind diese Änderungen ebenfalls rückgängig zu machen, wobei wiederum zu prüfen ist, dass sich der Quellknoten der Transition noch im Zustand Untouched befindet.

- *delay-Kontroll-Aktion:*
  Wurde ein Knoten aus der Adaptations-Region, der nicht mehr innerhalb des Gültigkeitsintervalls zur Ausführung kommt, durch Umsortieren verzögert, muss der Knoten wieder an seiner ursprünglichen Position eingefügt werden, wenn der Kontrollfluss diese noch nicht erreicht hat. Ist der Kontrollfluss schon über diese Stelle hinweg, kann der Knoten an der aktuellen Unterbrechungsstelle eingefügt werden, wenn keine Regeln verletzt werden. Ansonsten muss der Knoten an seiner neuen Position bleiben und die *delay*-Kontroll-Aktion kann nicht rückgängig gemacht werden.
  Wurde zur Verzögerung eine Split/Join-Region konstruiert (vgl. Abschnitt 5.1.5) so muss diese gelöscht und die ursprünglich sequentielle Ausführung des zu verzögernden Knotens und seiner Nachfolger wiederhergestellt werden. Dazu sind der Split- und der Join-Knoten sowie zusätzlich eingefügte Transitionen zu löschen und die übrigen Kanten so umzusetzen, dass die Knoten wieder in einer Sequenz nacheinander ausgeführt

werden. Anschließend ist noch die verzögernde Transitionsbedingung an der eingehenden Kontrollfluss-Transition des geänderten Knotens zu löschen bzw. durch eine eventuell vorher vorhandene zu ersetzen.

Ist der Kontrollfluss bereits über den Split-Knoten hinweg, kann die Verzögerung nicht rückgängig gemacht werden.

- *add-Kontroll-Aktion:*

  Galt für die Adaptations-Region eine *add*-Kontroll-Aktion, wurde ein Knoten neu eingefügt. Falls dieser bis zu einem Zeitpunkt $T' < T_{End}$, mit $(T_{End} - T') =$ *Ausführungsdauer der neuen Aktivität*, nicht ausgeführt wurde, veranlasst der Workflow-Monitoring-Agent eine Unterbrechung der Instanz und beauftragt den Adaptations-Agenten, den neu eingefügten Knoten von seiner jetzigen Stelle zu löschen und direkt an der aktuellen Unterbrechungsstelle einzufügen (vgl. Abschnitt 4.3). Die Adaptation wird also nicht rückgängig gemacht, sondern die Einfügestelle wird verschoben, damit die neue Aktivität auf jeden Fall ausgeführt wird.

  Wurde der Knoten in eine bestehende Sequenz eingefügt, wird er von dort gelöscht und an der aktuellen Unterbrechungsstelle neu eingefügt.

  Wurde er über einen Parallelpfad eingefügt, müssen zuerst die neu konstruierte Split/ Join-Region und der neu eingefügte Aktivitätsknoten gelöscht werden. Anschließend wird der Knoten wie in Abschnitt 5.1.6 beschrieben an der aktuellen Unterbrechungsstelle in die bestehende Sequenz eingefügt, so dass er bei der weiteren Ausführung der Instanz noch innerhalb des Gültigkeitsintervalls zur Ausführung kommt. Sollte der Knoten an der aktuellen Unterbrechungsstelle nicht eingefügt werden können, weil dadurch Regeln verletzt würden, wird eine Nutzeranfrage generiert, über die eine Einfügestelle angegeben werden kann.

6. Implementierung

Um die Korrektheit der in Kapitel 4 und 5 entwickelten Algorithmen zur Bestimmung der Adaptations-Region und zur Adaptation einer Workflow-Instanz zu überprüfen, wurde eine prototypische Implementierung des Adaptations-Agenten erstellt. Der Agent wurde dabei nicht in das Gesamtsystem AGENTWORK eingebunden, sondern arbeitet unabhängig von den übrigen Komponenten wie der Workflow-Engine und den anderen Agenten.
Das Auftreten logischer Fehler, die Folgerung von Kontroll-Aktionen und die Unterbrechung einer Workflow-Instanz an einer bestimmten Stelle werden somit nur simuliert, um die entwikkelten Algorithmen zu testen.
Im Folgenden wird zuerst das Simulationskonzept genauer erläutert (Abschnitt 6.1). Anschließend wird die Implementierung vorgestellt (Abschnitt 6.2), die mit Visual C++, Version 6.0, unter Windows NT erfolgte. Als Datenbanksystem wurde IBM db2 in den Versionen 5.2 bzw. 6.1 verwendet. Zum Schluss werden die Ergebnisse der Tests zusammengefasst (Abschnitt 6.3).

6.1 Simulationskonzept

Der Test des Adaptations-Agenten erfolgt unabhängig vom Gesamtsystem. Deshalb müssen die Schritte, die zum Aufruf des Agenten führen, simuliert werden. In diesem Abschnitt wird erläutert, welche Anforderungen an eine solche Simulation gestellt werden.

Um aussagefähige Testergebnisse zu erhalten, muss mit einer Menge von beliebigen Workflows gearbeitet werden. Beliebig bedeutet, dass die Kontroll-Regionen, aus denen der Workflow aufgebaut ist, zufällig aufeinander folgen und ineinander geschachtelt sein können. Auch die Knotentypen (Aktivitäts- oder Kommunikationsknoten) und Aktivitätstypen werden zufällig zugewiesen. Damit wird sichergestellt, dass bei der Definition der Workflows kein Wissen darüber einfließt, wie die Algorithmen arbeiten.
Kontroll-Aktionen sind die Arbeitsanweisungen für den Adaptations-Agenten und werden im Gesamtsystem durch den Ereignis-Agenten gefolgert. In der Simulation werden sie manuell oder automatisch erzeugt. Dabei ist wichtig, dass nicht nur einzelne Kontroll-Aktionen erzeugt

werden, sondern auch zufällige Kombinationen von mehreren. Diese sollten dann auch unterschiedliche Gültigkeitsintervalle haben, damit beliebig komplexe Fälle getestet werden können.
Außerdem muss noch die Unterbrechung eines Workflows simuliert werden. Dazu wird die Unterbrechungsmenge U bestimmt, an der die Ausführung eines Workflows angehalten wurde, um ihn zu adaptieren. Um den unterbrochenen Workflow wirklich in einen Zustand zu bringen, wie er auch bei der realen Ausführung auftritt, müssen die Knoten- und Transitionszustände entsprechend gesetzt werden. Die Knoten und Transitionen vor der Unterbrechungsstelle sind im Zustand Committed oder Unreachable, die danach im Zustand Untouched oder Unreachable. Der Zustand Unreachable wird von den nicht ausgewählten Pfaden einer OR-Verzweigung angenommen. Die Knoten, die den logischen Fehler ausgelöst haben, sind im Zustand Logically-Failed. Die Knoten, deren Ausführung unterbrochen wurde, sind in einem der Zustände Control-Activated, Object-Activated oder Currently-Processed. Transitionen, deren Auswertung unterbrochen wurde, befinden sich im Zustand Currently-Evaluated.

Workflows, Kontroll-Aktionen, die Unterbrechungsmenge und die korrekte Zuordnung der Knoten- und Transitionszustände müssen also simuliert werden. In der implementierten Testumgebung werden die Kontroll-Aktionen und die Unterbrechungsmenge über eine grafische Oberfläche von Hand erzeugt. Die Knoten- und Transitionszustände werden direkt in der Datenbank, die die Workflows enthält, ebenfalls manuell zugeordnet. Alle Schritte können automatisiert werden, wobei als weitere Simulationsstufe auch die Auswahl der zu ändernden Workflows automatisch erfolgen sollte.
Die Erzeugung der Workflows ist bereits automatisiert. Durch einen Graph-Generator, der als Eingabeparameter die gewünschte Anzahl der Schleifen und Split/Join-Regionen im zu erzeugenden Workflow erhält, werden beliebige Workflows generiert, die die oben angeführten Bedingungen erfüllen. Allerdings wird nur der Kontrollfluss erzeugt, da der Prototyp des Adaptations-Agenten den Datenfluss vernachlässigt. Synchronisations-Transitionen gehören zwar zum Kontrollfluss, werden aber in der aktuellen Version des Graph-Generators nicht berücksichtigt und müssen von Hand nachträglich eingefügt werden.

6.2 Realisierung

Die Implementierung des Agenten besteht aus zwei Teilen: zum einen dem Entwurf und die Implementierung der benötigten Datenbanken, zum anderen die Implementierung der Algorithmen.

Der Adaptations-Agent benötigt zwei Datenbanken: die Runtime-Datenbank, in der die Workflow-Definitionen abgelegt sind und mit der auch der Graph-Generator arbeitet, und eine eigene Agenten-Datenbank, in der Informationen zu den Kontroll-Aktionen, den Adaptations-Regionen und den Adaptationen gespeichert werden.
Da der Prototyp des Adaptations-Agenten unabhängig vom Gesamtsystem implementiert und getestet wurde, wird nicht die Runtime-Datenbank des Gesamtsystems (vgl. [Die00]) verwendet sondern eine verkleinerte Variante. Diese enthält nur die Tabellen, die nötig sind, um den Kontrollfluss einer Workflow-Instanz, bestehend aus Knoten und Transitionen unterschiedlichen Typs, zu speichern. Auf den Datenfluss wurde verzichtet, um den Aufwand für die Implementierung des Prototypen in einem vertretbaren Rahmen zu halten. Das Schema dieser relationalen Datenbank ist im Anhang A.1 abgebildet und wird hier kurz erläutert.

- Die Tabelle WORKFLOW enthält für jede Workflow-Instanz eine eindeutige Bezeichnung (ID) und einen Namen (Name).

- In der Tabelle NODE werden die Informationen zu den Knoten des Workflows, in der Tabelle TRANSITION die zu den Transitionen des Workflows gespeichert.

- Aktivitätsknoten werden bestimmte Aktivitätstypen zugeordnet. Alle Aktivitätstypen, die in den Workflows vorkommen können, werden in der Tabelle ACTIVITY_DEF abgelegt und den Knoten über eine Fremdschlüsselbeziehung zugeordnet.

- Eine Knoten kann eine bestimmte Ausführungsdauer haben, die durch einen Zeitwert, bestehend aus Wert und Einheit, abgebildet wird. Auch temporale Transitions-Bedingungen werden über solche Zeitwerte abgebildet. Um für Knoten oder Transitionen, die keinen Zeitwert besitzen, nicht vier Nullwerte (für minimalen, maximalen oder

durchschnittlichen Wert und die Einheit) speichern zu müssen sondern nur einen, werden alle vorkommenden Zeitwerte in der Tabelle DURATION abgelegt und über eine Fremdschlüsselbeziehung den Einträgen in den Tabellen NODE bzw. TRANSITION zugeordnet.

Die ebenfalls relationale Agenten-Datenbank besteht aus fünf Tabellen (vgl. Schema im Anhang A.2).

- In der Tabelle COMPLETED_ACTIONS werden alle Kontroll-Aktionen gespeichert, die der Adaptations-Agent bearbeitet hat. Dabei wird zu jeder Kontroll-Aktion die eindeutige Bezeichnung der Instanz abgelegt, für die die Kontroll-Aktion gefolgert wurde.

- Die beiden Tabellen ADAPT_REG_NODES und ADAPT_REG_TRANS enthalten die Knoten und Transitionen der vom Adaptations-Agenten berechneten Adaptations-Regionen. Zusätzlich zu den Informationen, die auch in der Runtime-Datenbank gespeichert sind, wie die ID, der Typ, der Aktivitätstyp usw. werden hier noch Werte abgelegt, die der Adaptations-Agent während der Bestimmung der Adaptations-Region und der Durchführung der Adaptationen erhält und für die weiteren Berechnungen benötigt. So wird z. B. die Ausführungsdauer des untersuchten Pfades bis zu diesem Knoten in *PathDuration* gespeichert und *NodeAdapted* gibt an, ob ein Knoten von einer Kontroll-Aktion betroffen war und geändert wurde.
  Wenn ein Knoten oder eine Transition aus einer Adaptations-Region von einer Kontroll-Aktion betroffen ist und geändert wird, wird in *AdaptationAction* die ID, unter der diese Kontroll-Aktion in der Tabelle COMPLETED_ACTIONS abgelegt ist, eingetragen.

- Die Tabelle ADDED_NODES enthält die IDs der Aktivitätsknoten, die durch *add*-Kontroll-Aktionen neu eingefügt wurden.

- In der Tabelle MONITORING_NODES wird für jeden neu eingefügten Monitoring-Knoten abgelegt, zu welcher Adaptations-Region, welcher Workflow-Instanz und welcher Kontroll-Aktion er gehört. Diese Informationen benötigt beispielsweise der Workflow-Monitoring-Agent, der die weitere Ausführung einer adaptierten Instanz überwacht,

um beim Erreichen eines Monitoring-Knotens die richtigen Schritte zu folgern (vgl. Abschnitt 4.3).

Der zweite Teil der Implementierung umfasst die Umsetzung der konzeptionell entwickelten Algorithmen in ein Programm. Wie oben schon erläutert, wurde dabei nur der Kontrollfluss einer Workflow-Instanz berücksichtigt, nicht aber der Datenfluss.
Für die Algorithmen zur Bestimmung der Adaptations-Region (vgl. Kapitel 4) stellt das keine Einschränkung dar, da der Datenfluss bei der Bestimmung der Adaptations-Region nur an einer einzigen Stelle berücksichtigt wird und zwar dann, wenn zwischen zwei Knoten in parallelen Pfaden einer Split/Join-Region eine Objektfluss-Transition existiert. Diese Transition wirkt, wie in Abschnitt 4.2.2.5 erläutert, wie eine Synchronisations-Transition und wird auch wie eine solche behandelt. Für Objektfluss-Transitionen zu konditionalen Transitionen wird angenommen, dass die Daten sofort zur Verfügung stehen, wenn der Kontrollfluss die Transition erreicht. Es tritt also keine zusätzliche Verzögerung ein, die bei der Bestimmung der Adaptations-Region berücksichtigt werden müsste. Durch das Auftreten von Objektfluss-Transitionen entsteht also keine neue Komplexität in den Algorithmen.
Von den Algorithmen für die Adaptation einer Workflow-Instanz wurden nur die Teile realisiert, die den Kontrollfluss betreffen. Diese Einschränkung verschlechtert die Aussagefähigkeit der Tests nicht wesentlich, da der Datenfluss meist in Folge von Änderungen am Kontrollfluss geändert werden muss.
Sehr komplexe Adaptations-Operationen wie *delay* und *add* wurden nur in ihrer Grundform implementiert, um zu testen, ob der gewählte Ansatz richtig ist. Die vollständige Implementierung wäre im Rahmen dieser Arbeit zu aufwändig gewesen.

Der Prototyp des Adaptations-Agenten verfügt über eine dialogbasierte Oberfläche, über die die ID der Workflow-Instanz, die adaptiert werden soll, die Knoten-IDs der Unterbrechungsmenge und die gefolgerten Kontroll-Aktionen eingegeben werden. Alternativ kann auch die ID einer in der Agenten-Datenbank gespeicherten Kontroll-Aktion eingegeben werden, für die die vorgenommenen Adaptationen rückgängig gemacht werden sollen (siehe Abbildung 6-1).

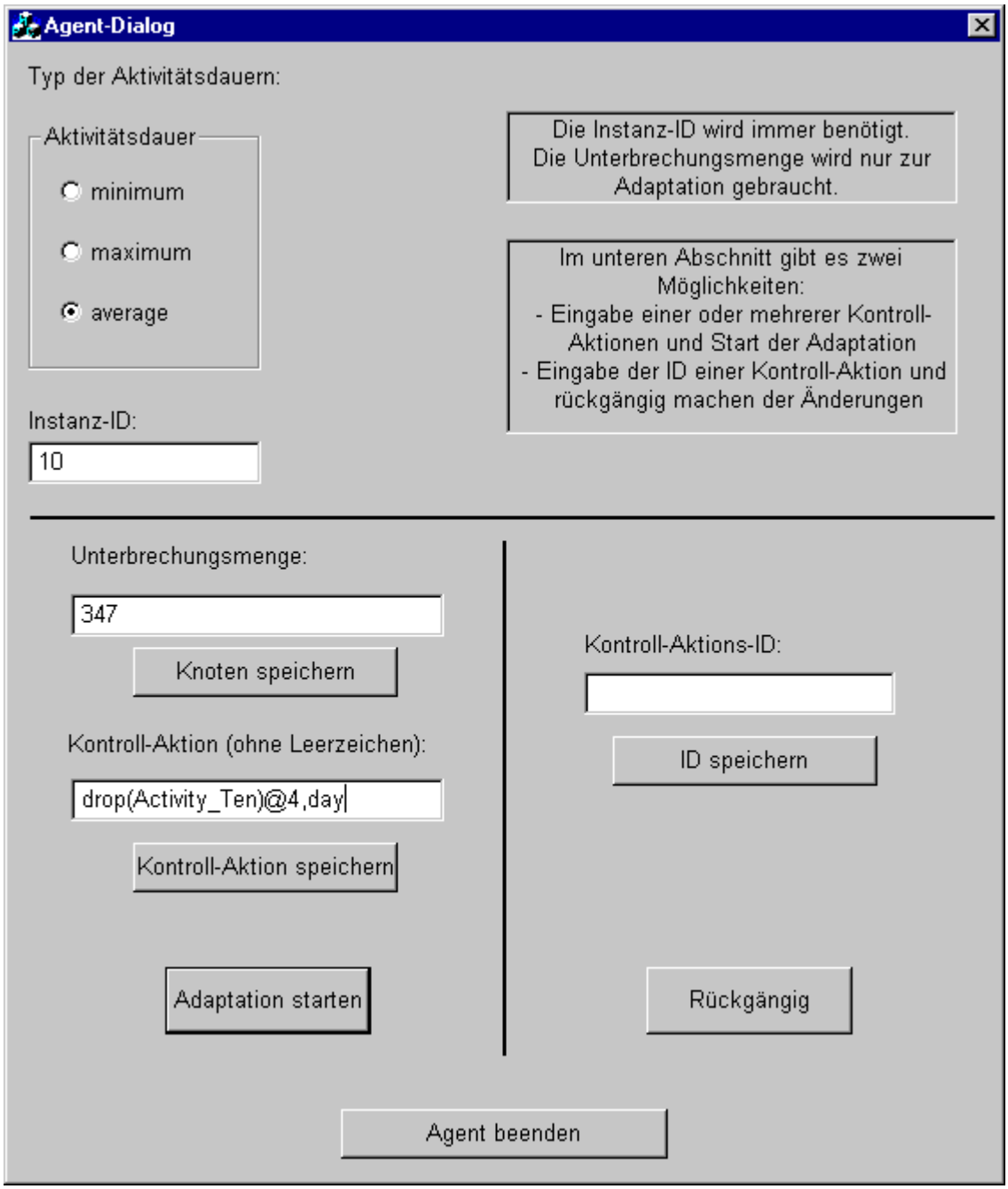


Abb. 6-1: Startdialog des Adaptations-Agenten.

Hat der Adaptations-Agent seine Berechnungen beendet und die Änderungen durchgeführt, werden wiederum über einen Dialog die Knoten der Adaptations-Region und die geänderten Knoten in Textform, nicht als Graph dargestellt (siehe Abbildung 6-2). Dabei werden für die Knoten der Adaptations-Region die Knoten-ID, der Knoten-Typ und, falls es ein Aktivitäts-

knoten ist, der Aktivitätstyp ausgegeben. Zu den geänderten Knoten werden die ID und die Kontroll-Aktion, die die Änderung verursachte, ausgegeben.

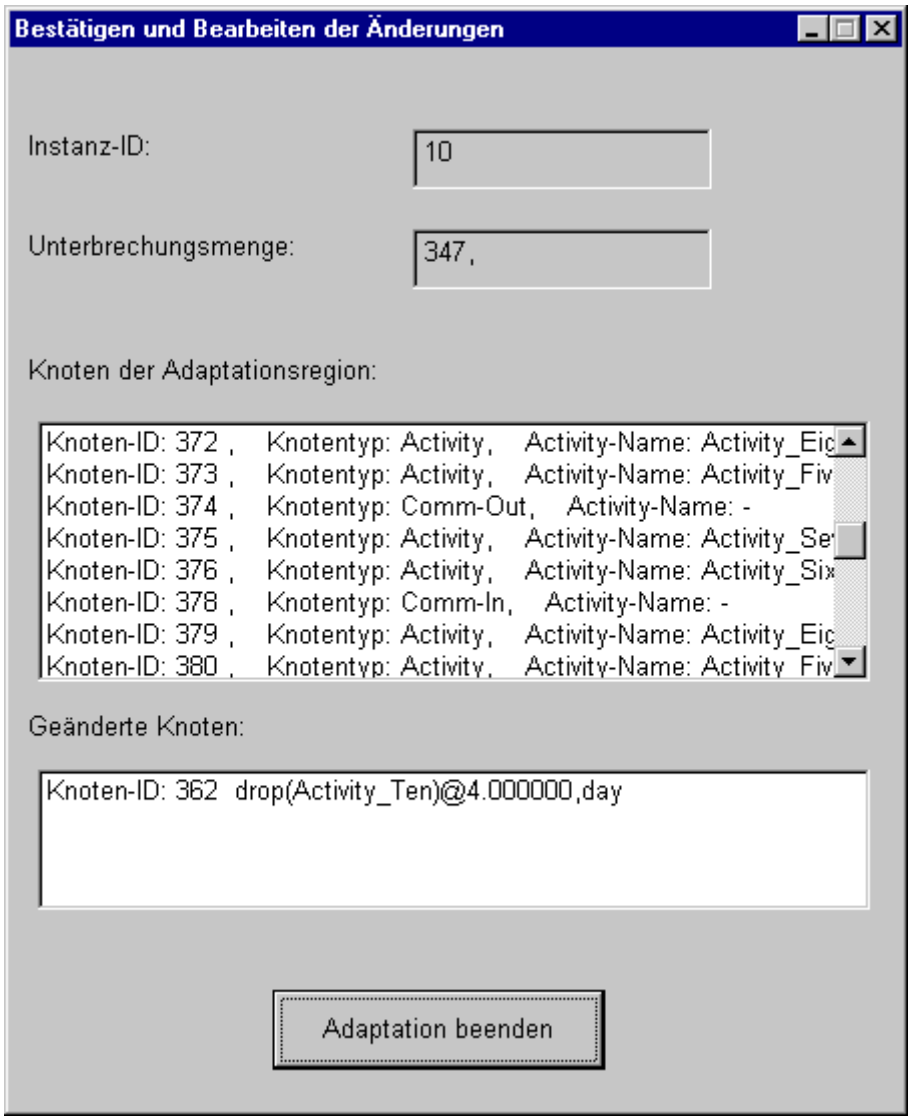


Abb. 6-2: Ausgabe der Adaptations-Region und der geänderten Knoten.

Die konzeptionell vorgesehene, abschließende Überarbeitung der an einer Workflow-Instanz vorgenommenen Änderungen durch einen autorisierten Nutzer, ist noch nicht implementiert. Sie soll ähnlich wie im Workflow-Editor (vgl. [Boe00]) über eine grafische Oberfläche erfolgen, in der der veränderte Workflow-Graph dargestellt wird und dann mit bestimmten Operatoren bearbeitet werden kann.

Wenn der Adaptations-Agent in das Gesamtsystem eingebunden wird, soll dessen Workflow-Editor für Teile dieser Aufgabe, wie die Anzeige des Workflows und die Anwendung der Operatoren, verwendet werden. Da die Einbindung aber in der vorliegenden Arbeit nicht vorgesehen ist und eine doppelte Implementierung der grafischen Oberfläche und der Operatoren zu aufwändig und nicht sinnvoll wäre, wird darauf verzichtet.

6.3 Ergebnisse

Mit der oben beschriebenen prototypischen Implementierung des Adaptations-Agenten wurden die in der vorliegenden Arbeit entwickelten Algorithmen auf ihre Korrektheit und Verwendbarkeit im Workflow-System AGENTWORK hin überprüft. Dabei wurden folgende Punkte getestet:

- Korrekte Berechnung der Adaptations-Region für eine gültige Kontroll-Aktion je Workflow-Instanz bzw. für mehrere gleichzeitig gültige Kontroll-Aktionen mit dem gleichen Gültigkeitsintervall. Dabei sind die Ausführungsdauern und Knotenmengen der einzelnen Kontroll-Regionen, aus denen ein Pfad aufgebaut ist, zu bestimmen und Synchronisations-Transitionen zwischen beliebigen parallelen Pfaden (auch zwischen unterschiedlichen Split/Join-Regionen) zu berücksichtigen (vgl. Kapitel 4).
- Korrekte Berechnung der Adaptations-Regionen, falls für eine Workflow-Instanz gleichzeitig mehrere Kontroll-Aktionen mit unterschiedlichem Gültigkeitsintervall gelten. In diesem Fall ist insbesondere eine Transformation der Gültigkeitsintervalle der einzelnen Kontroll-Aktionen nötig Diese erfolgt so, dass möglichst wenige Adaptations-Regionen berechnet werden müssen (vgl. Abschnitt 3-2).
- Korrekte Durchführung der Kontrollfluss-Adaptationen. Hierbei sind Knoten und Transitionen zu löschen, zu verschieben oder neu einzufügen. Möglicherweise müssen auch neue Kontroll-Regionen (beispielsweise Split/Join-Regionen bei *add*- oder *delay*-Kontroll-Aktionen) eingefügt werden (vgl. Kapitel 5).
- Fehlerfreie Rücknahme von Adaptationen. Insbesondere sind gelöschte oder verschobene Knoten wieder an der richtigen Position einzufügen, und neu eingefügte Knoten und Kontroll-Konstrukte (z. B. Split/Join-Regionen bei *add*- oder *delay*-Kontroll-Aktionen) wieder vollständig zu entfernen (vgl. Kapitel 5).

Die Auswertung der Ergebnisse erfolgte dabei über den in Abbildung 6-2 gezeigten Dialog, in dem die Knoten der Adaptations-Region und die durch Kontroll-Aktionen geänderten Knoten in Textform ausgegeben werden. Die Korrektheit der Berechnungen und Änderungen wurde anhand der Einträge in der Datenbank geprüft, da, wie am Ende des letzten Abschnitts erläutert, für den Adaptations-Agenten noch keine grafische Oberfläche zur Anzeige eines Workflow-Graphen zur Verfügung steht.
Während der Testphase mussten der Algorithmus zur Rücknahme von Änderungen und die Agenten-Datenbank mehrfach erweitert und angepasst werden, damit alle benötigten Informationen für diese Operation zur Verfügung stehen. Die Algorithmen zur Bestimmung der Adaptations-Region mussten nur geringfügig geändert werden, wobei meistens nur Hilfsfunktionen (z. B. zur Bestimmung bestimmter Knotenmengen) betroffen waren. Ähnlich war es auch bei den Algorithmen zur Durchführung der Änderungen, wobei bei diesen die Implementierung später noch erweitert werden muss, damit auch der Datenfluss korrekt adaptiert wird und alle Kontroll-Aktionen in vollem Umfang implementiert sind.

Nicht implementiert und getestet wurde die Behandlung von Knoten, die durch die Änderungen aus der Adaptations-Region herausgeschoben werden oder eventuell früher ausgeführt werden können (vgl. Abschnitt 5.2). Insbesondere die Bestimmung von Knoten, die möglicherweise früher ausgeführt werden können, ist, wie im obigen Abschnitt dargestellt, eine relativ komplexe Aufgabe. Das liegt vor allem daran, dass der erste Knoten außerhalb der Adaptations-Region von einer Kontroll-Aktion betroffen sein kann, seine Nachfolger aber nicht mehr. Eine frühere Ausführung dieser Nachfolgerknoten würde zu einer Umstrukturierung des Workflows führen, die nur mit zusätzlichem Wissen (z. B. über die Reihenfolge oder Abhängigkeit bestimmter Aktivitäten) durchgeführt werden kann. Diese Informationen müssen in der Wissensbasis abgelegt werden, die aber noch nicht implementiert ist.
Auch die Grenzen des Adaptations-Agenten bzgl. Anzahl und Typ gleichzeitig gültiger Kontroll-Aktionen wurden nicht erschöpfend ausgetestet, da die bisherige einfache Simulationsumgebung solche Dinge nur unzureichend unterstützt.

Zusammenfassend kann man sagen, dass die in der vorliegenden Arbeit entwickelten Algorithmen zur Adaptation einer Workflow-Instanz die Anforderungen (Bestimmung der Adaptations-Region, Durchführung von Änderungen, Rücknahme von Änderungen) erfüllen. Der

Adaptations-Agent kann also in einer erweiterten Form im System AGENTWORK verwendet werden, um Workflow-Instanzen, die durch das Auftreten logischer Fehler nicht mehr adäquat sind, durch Adaptationen des Kontroll- und Datenflusses an die neue Situation anzupassen. Die Erweiterungen umfassen dabei vorrangig die noch nicht implementierten Teile wie z. B. die Behandlung des Datenflusses, Überarbeitung der Änderungen durch den Nutzer über eine grafische Oberfläche und die Behandlung von Knoten, die durch die Durchführung der Änderungen aus der Adaptations-Region herausgeschoben wurden oder möglicherweise früher ausgeführt werden können. Zusätzlich sind noch Schnittstellen zu den übrigen Komponenten des Systems (z. B. gemeinsame Datenbank, Workflow-Engine und -Editor, die übrigen Agenten) zu entwickeln.

7. Zusammenfassung

7.1 Zusammenfassung der Ergebnisse

In der vorliegenden Arbeit wurde der Adaptations-Agent, eine Komponente des Workflow-Systems AGENTWORK, konzeptionell entwickelt und prototypisch implementiert. Der Agent ist im Gesamtsystem für die Adaptation von Workflow-Instanzen verantwortlich, deren Kontrollfluss durch das Auftreten eines logischen Fehlers nicht mehr adäquat ist. Welche Adaptationen vorzunehmen sind, erfährt der Agent dabei über Kontroll-Aktionen, die für ein bestimmtes Zeitintervall *T* gültig sind.
Der Agent muss also zuerst den Teil der Workflow-Instanz bestimmen, der voraussichtlich innerhalb des Gültigkeitsintervalls *T* durchlaufen werden wird. Dieser Teilgraph des Workflow-Graphen bildet die Adaptations-Region. Die Knoten aus der Adaptations-Region, die von einer Kontroll-Aktion betroffen sind, müssen geändert werden.
Abhängig vom Typ einer Kontroll-Aktion ist ein betroffener Knoten zu löschen, zu verzögern, oder neu einzufügen. Möglicherweise müssen bei der Bearbeitung einer Kontroll-Aktion weitere Knoten des Workflows geändert werden oder es sind zusätzliche Kontrollknoten einzufügen (z. B. bei einer *add*-Kontroll-Aktion). Auch die ein- und ausgehenden Transitionen des oder der betroffenen Knoten sind zu löschen oder umzusetzen, so dass die Workflow-Instanz nach den Änderungen wieder aus korrekten Kontroll-Regionen besteht.

Durch die prototypische Implementierung sollte gezeigt werden, dass die entwickelten Algorithmen korrekt arbeiten und für den Einsatz im Workflow-System AGENTWORK geeignet sind. Da die Implementierung unabhängig vom Gesamtsystem erfolgte, wurde das Auftreten logischer Fehler, die Folgerung von Kontroll-Aktionen und die Unterbrechung von Workflow-Instanzen simuliert. Die Tests haben gezeigt, dass die Algorithmen korrekt und für den Einsatz im System AGENTWORK geeignet sind. Dabei wurde überprüft, ob die Adaptations-Regionen richtig berechnet und die Änderungen am Kontrollfluss korrekt durchgeführt werden. Außerdem wurde getestet, ob Änderungen auch wieder rückgängig gemacht werden können.
Weitergehende Tests, die eventuelle Fehler oder Lücken im Modell des Gesamtsystems AGENTWORK aufdecken könnten, wurden nicht durchgeführt, da dafür eine komplexere Simulationsumgebung nötig wäre, als sie im Rahmen dieser Arbeit entwickelt werden konnte.

7.2 Ausblick

Die weitere Entwicklung des Adaptations-Agenten kann in zwei Richtungen fortgesetzt werden. Der Agent kann in das Workflow-System AGENTWORK integriert werden oder in einer erweiterten Simulationsumgebung einschließlich des zugrunde liegenden Modells (vgl. [Mue00]) umfassend getestet werden.

Wenn der Agent in das Gesamtsystem integriert werden soll, muss die bisherige Implementierung um die Bearbeitung des Datenflusses, die abschließende Überarbeitung einer geänderten Workflow-Instanz durch einen Nutzer (mit Hilfe des Workflow-Editors des Gesamtsystems) und die Behandlung von möglicherweise früher auszuführenden oder nicht mehr im Gültigkeitsintervall ausführbaren Knoten erweitert werden. Außerdem sind die Algorithmen für die komplexen Kontroll-Aktionen (z. B. *add* oder *delay*) zu erweitern. Die Datenbankschnittstelle zur verkleinerten Runtime-Datenbank ist durch die Schnittstelle zur Runtime-Datenbank des Gesamtsystems zu ersetzen. Zusätzlich sind Schnittstellen zu den übrigen Systemkomponenten wie der Workflow-Engine, dem Workflow-Editor und den anderen Agenten zu konzipieren und zu implementieren.

Die Alternative zur Einbindung des Agenten in das Gesamtsystem wäre, die Simulationsumgebung so zu erweitern, dass damit das Auftreten logischer Fehler, die Folgerung von Kontroll-Aktionen, die Unterbrechung von Instanzen und der Aufruf des Adaptations-Agenten simuliert werden können. Auch erneute Unterbrechungen einer geänderten Workflow-Instanz, um Änderungen nachzuholen oder rückgängig zu machen, sollten möglich sein. Damit könnten Lücken im Gesamtmodell, wie z. B. unnötige oder bisher nicht berücksichtigte Kontroll-Aktionen, entdeckt werden, bevor das System in den laufenden Betrieb übergeht.
Mit einer solchen Simulationsumgebung könnten auch Abhängigkeiten zwischen den einzelnen Kontroll-Aktionen, die bis jetzt noch nicht berücksichtigt wurden, entdeckt werden. Wenn sich z. B. herausstellt, dass bei einer bestimmten Kombination von gleichzeitig gültigen Kontroll-Aktionen sehr häufig nachadaptiert werden muss, kann das bereits bei der Folgerung der Kontroll-Aktionen berücksichtigt werden.

Der Simulator müsste zusätzlich um eine grafische Oberfläche und Tools zur automatischen Auswertung (beispielsweise die Bestimmung der Anzahl der Nachadaptationen) erweitert werden, die die Arbeit erleichtern.

Sinnvoll wäre eine Kombination aus beiden Möglichkeiten. Zuerst sollte der Adaptations-Agent erweitert (z. B. Datenfluss, Überarbeitung der Änderungen) und zusammen mit weiteren Teilen des Systems AGENTWORK in einer Simulationsumgebung getestet werden. Parallel dazu wird er in das Gesamtsystem eingefügt, um Probleme, die möglicherweise erst beim Betrieb des Gesamtsystems auftreten, frühzeitig erkennen und beheben zu können.

Literaturverzeichnis

[BEM99] Bradshaw, J.; Etzioni, O.; Müller, J. (1999). *AGENTS '99. Proceedings of the Third Annual Conference on Autonomous Agents*, May 1-5, 1999, Seattle, WA, USA. ACM Press.

[BM97] Bemmel, van J.H.; Musen, M. (Eds.) (1997): *Handbook of Medical Informatics*. Springer, Berlin.

[Boe00] Böhme, R. (2000): *Konzeption und prototypische Implementierung eines Workflow-Editors*. Diplomarbeit. Universität Leipzig.

[Ca98] Casati, F. (1998): *Models, Semantics and Formal Methods for the Design of Workflows and their Exceptions.* Ph.D. Thesis. Politecnico di Milano.

[Coo99] Proceedings of the Fourth IFCIS International Conference on Cooperative Information Systems (CoopIS'99), Edinburgh, Scotland, September 1999. IEEE Computer Press.

[Die98] Diehl, V.; Franklin, J.; Hasenclever, D.; Tesch, H.; Pfreundschuh, M.; Lathan, B.; Paulus, U.; Sieber, M.; Rueffer, J.U.; Sextro, M.; Engert, A.; Wolf, J.; Hermann, R.; Holmer, L.; Stappert-Jahn, U.; Winnerlein-Trump, E.; Wulf, G.; Krause, S.; Glunz, A.; von, Kalle, K.; Bischoff, H.; Haedicke, C.; Duehmke, E.; Georgii, A.; Loeffler, M. (1998): *BEACOPP, a new dose-escalated and accelerated regimen, is at least as effective as COPP/ABVD in patients with advanced-stage Hodgkin's lymphoma: interim report from a trial of the German Hodgkin's Lymphoma Study Group*. Journal of Clinical Oncology 16(12): S. 3810-21.

[Die00] Dietzsch, A. (2000): *Konzeption und prototypische Implementierung einer Workflow-Engine für dynamische Adaptation*. Diplomarbeit. Universität Leipzig.

[DK+98] Dogac, A.; Kalinichenko, L.; Ozsu, T.; Sheth, A. (Eds.) (1998): *Workflow Management Systems and Interoperability.* Springer, Berlin.

[El92] Elmagarmid, A.K. (Ed.) (1992): *Database Transaction Models for Advanced Applications.* Morgan Kaufmann Publishers, San Moteo, California.

[FG97] Franklin, S.P.; Graesser, A. (1997): *Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents.* In [MWJ97]: S. 21-35.

[GPW99] Georgakopoulos, D.; Prinz, W.; Wolf, A. (Eds.) (1999): *Proceedings of the International Joint Conference on Work Activities Coordination and Collaboration (WACC '99), San Francisco, February 1999.* ACM Press, New York.

[Ham90] Hammond, K. J. (1990): *Explaining and Repairing Plans That Fail*. Artificial Intelligence 45: S. 173-228.

[HH+99] Heinl, P.; Horn, S.; Jablonksi, S.; Neeb, J.; Stein, K.; Teschke, M. (1999): *A Comprehensive Approach to Flexibility in Workflow Management Systems*. In [GPW99]: S. 79-88.

[HR99] Härder, T.; Rahm, E (1999): *Datenbanksysteme - Konzepte und Techniken der Implementierung.* Springer-Verlag, Berlin.

[KG99] Kradolfer, M.; Geppert, A. (1999): *Dynamic Workflow Schema Evolution based on Workflow Type Versioning and Workflow Migration*. In [Coo99].

[Mic97] Michaelis, J. (1997): *Biostatistical Methods.* In [BM97]: S. 387-397.

[MR00] Müller, R., Rahm, E. (2000): *Dealing with Logical Failures for Collaborating Workflows*. In: Etzion, O., Scheuermann, P. (Eds.): Proceedings of Fifth International Conference on Cooperative Information Systems (CoopIS), Eilat, Israel, Sep. 2000.

[MS99] Meyer, J.-J.C., Schobbens, P.-Y. (Eds.) (1999): *Formal Models of Agents*. LNCS 1760, Springer, Berlin.

[MSR99] Müller, J.P.; Singh, M.P.; Rao, A.S. (Eds.) (1999): *Intelligent Agents V.* Proceedings of 5th International Workshop on Agent Theories, Architectures and Languages (ATAL98), Paris, France, July 1998. Springer, Berlin.

[Mue00] Müller, R. (2000): *Event-Oriented Dynamic Adaptation of Workflows. Model, Architecture and Implementation*. Dissertation. Universität Leipzig.

[MWJ97] Müller, J.P.; Wooldridge, M.J.; Jennings, N.R. (Eds.) (1997): *Intelligent Agents III. Agent Theories, Architectures, and Languages.* ECAI'96 Workshop, Budapest, Hungary, August 1996, Proceedings. Springer, Berlin.

[NHL94] Löffler, M., Pfreundschuh, M. (1994): *Integratives Konzept zur Behandlung hochmaligner Non-Hodgkin-Lymphome (Studie B)*. Homburg und Leipzig.

[OMG97] Object Management Group, Inc. (1997): CORBAservices: Common Object Services Specification. http://www.omg.org.

[RD97] Reichert, M., Dadam, P. (1998): *$ADEPT_{flex}$ - Supporting Dynamic Changes of Workflows Without Loosing Control*. Journal of Intelligent Information Systems 10: S. 93 - 129.

[VW98] Vossen, G., Weske, M. (1998): *The WASA Approach to Workflow Management for Scientifc Applications*. In [DK+98].

[Wäc96] Wächter, H. (1996): *Fehlertolerantes Workflow-Management*. Doctoral Thesis. Verlag Dr. Kovac.

[WfMC99] Workflow Management Coalition (1999): *Basic Terminology & Glossary*. Workflow Management Coalition, http://www.wfmc.org.

[$WJ95_A$] Wooldridge, M.; Jennings, N.R. (1995): *Agent Theories, Architectures, and Languages: a Survey.* In [$WJ95_B$]: S. 1 - 22.

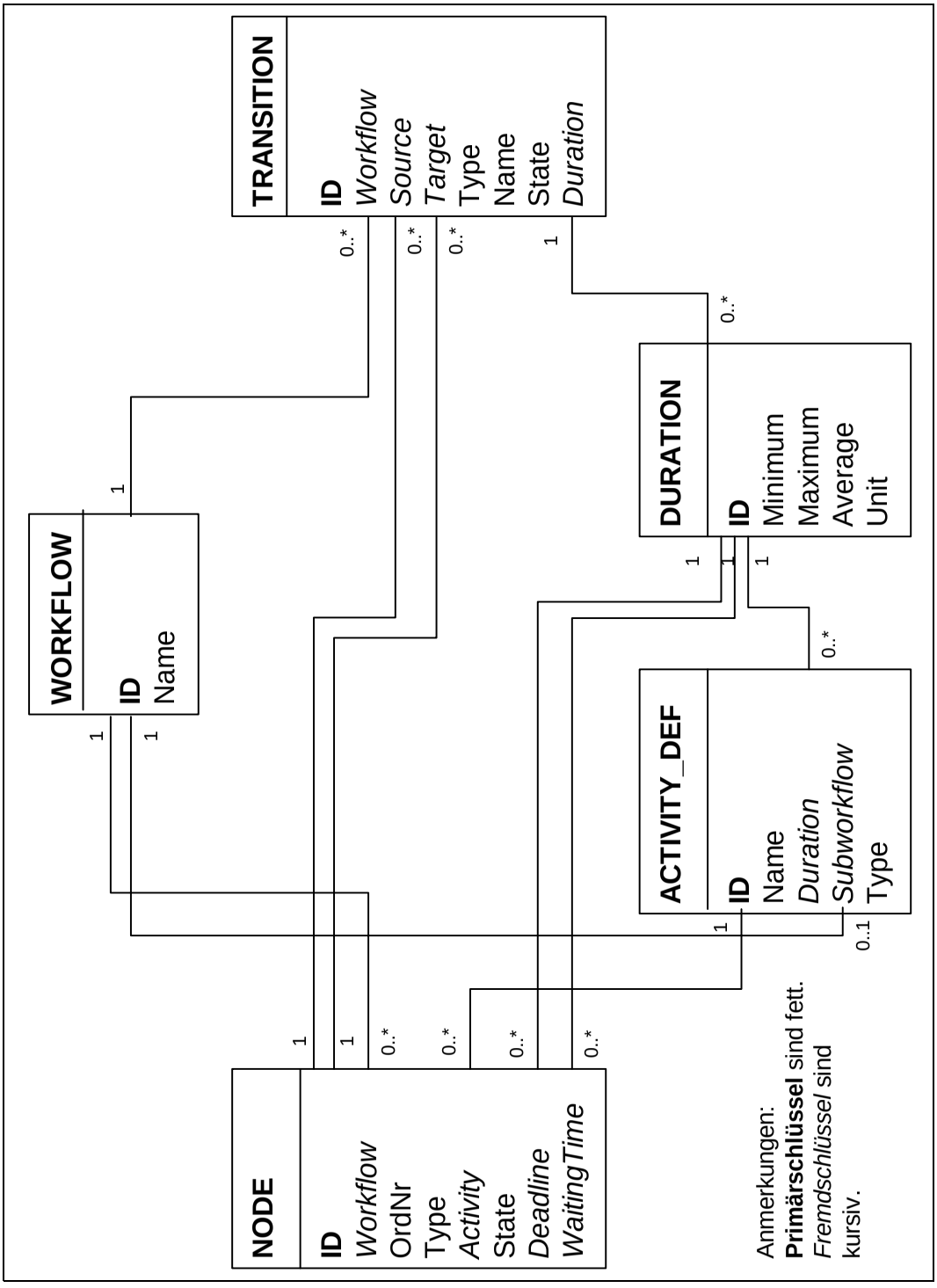
[$WJ95_B$] Wooldridge, M.J.; Jennings, N.R. (Eds.) (1995): *Intelligent Agents. ECAI-94 Workshop on Agent Theories, Architectures, and Languages.* Amsterdam, The Netherlands, August 1994. Proceedings. Springer, Berlin.

Abbildungsverzeichnis

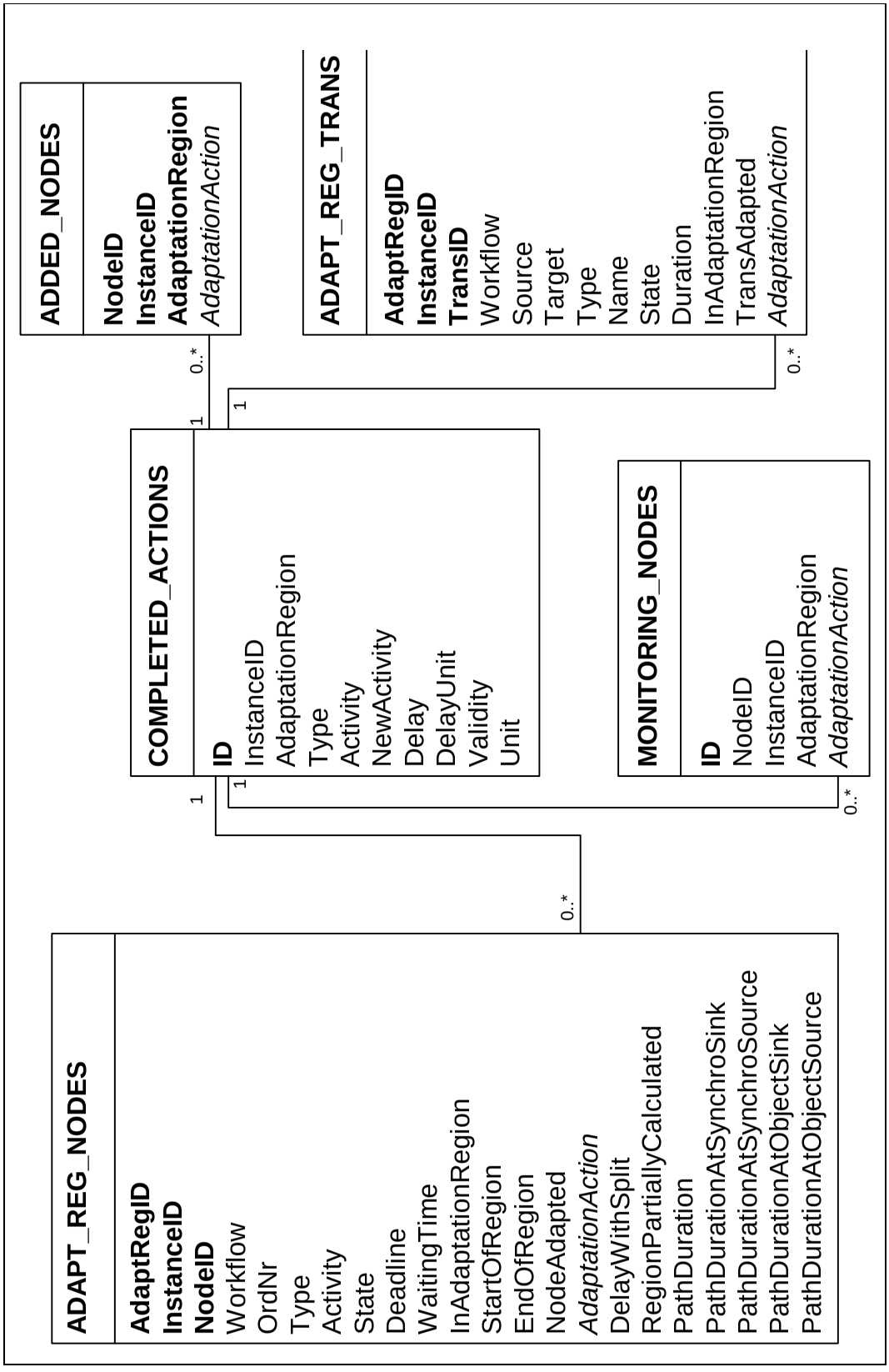
Tabellenverzeichnis

Anhang A: Datenbankschemata

A.1 Verkleinerte Runtime-Datenbank

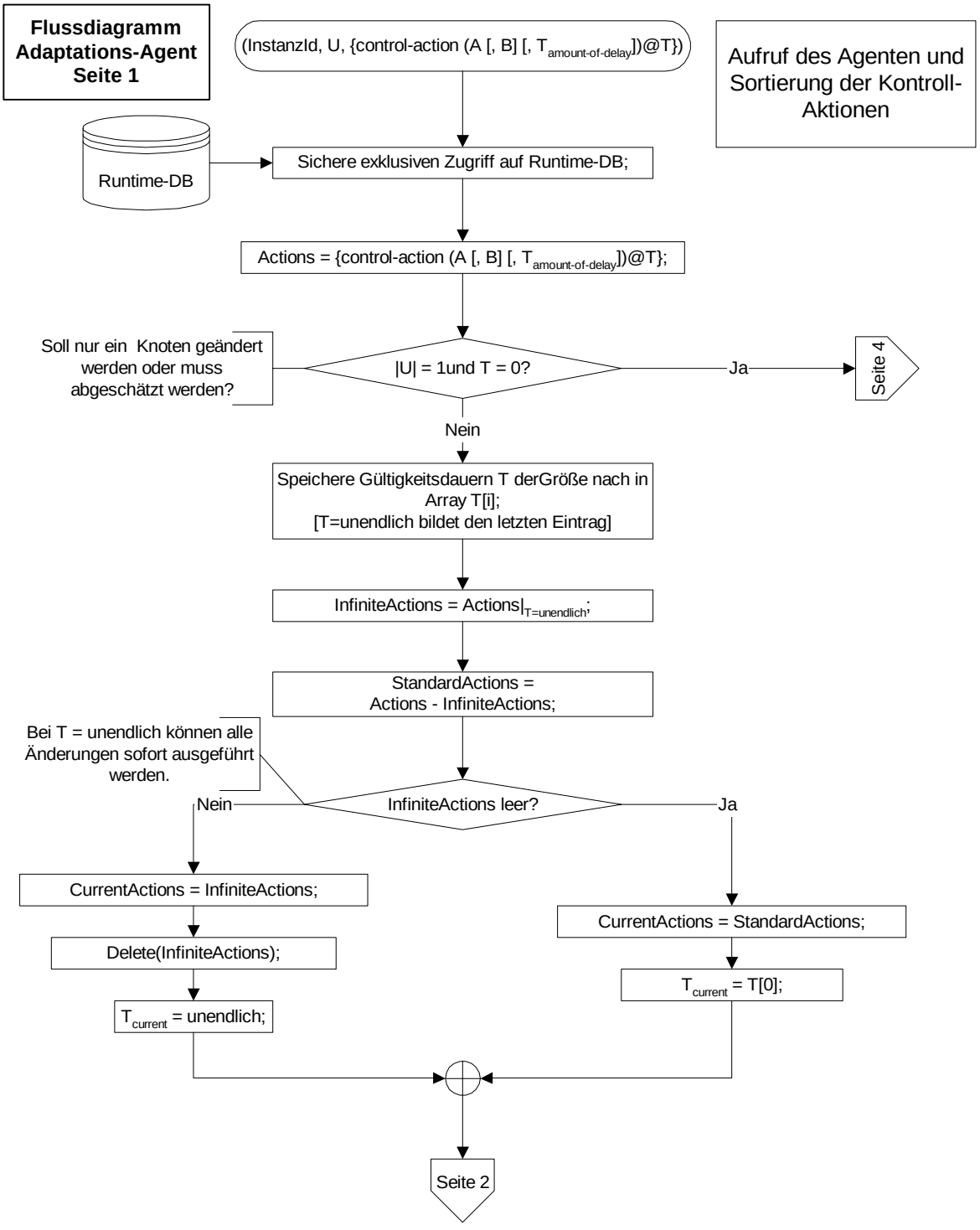


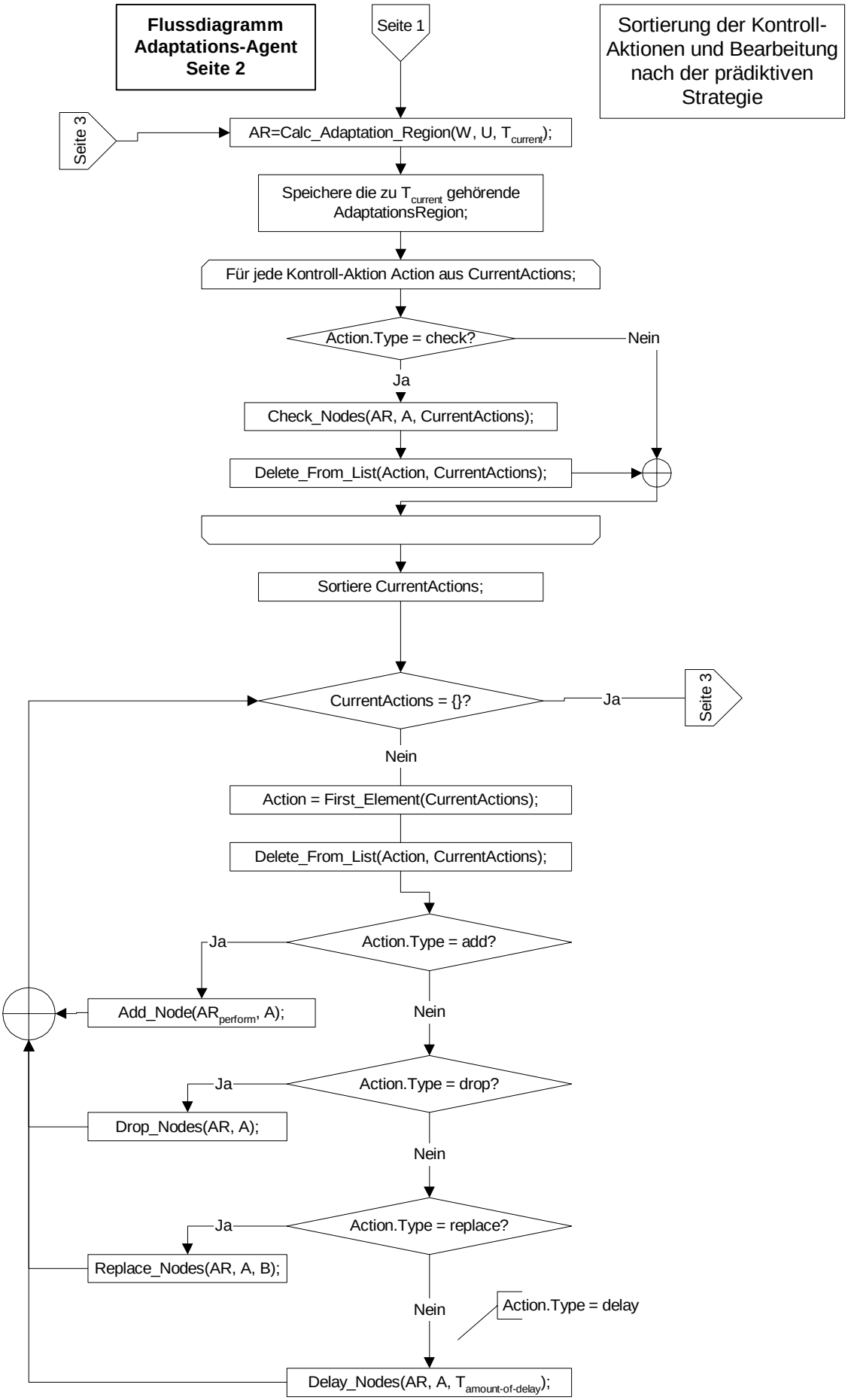
A.2 Agenten-Datenbank

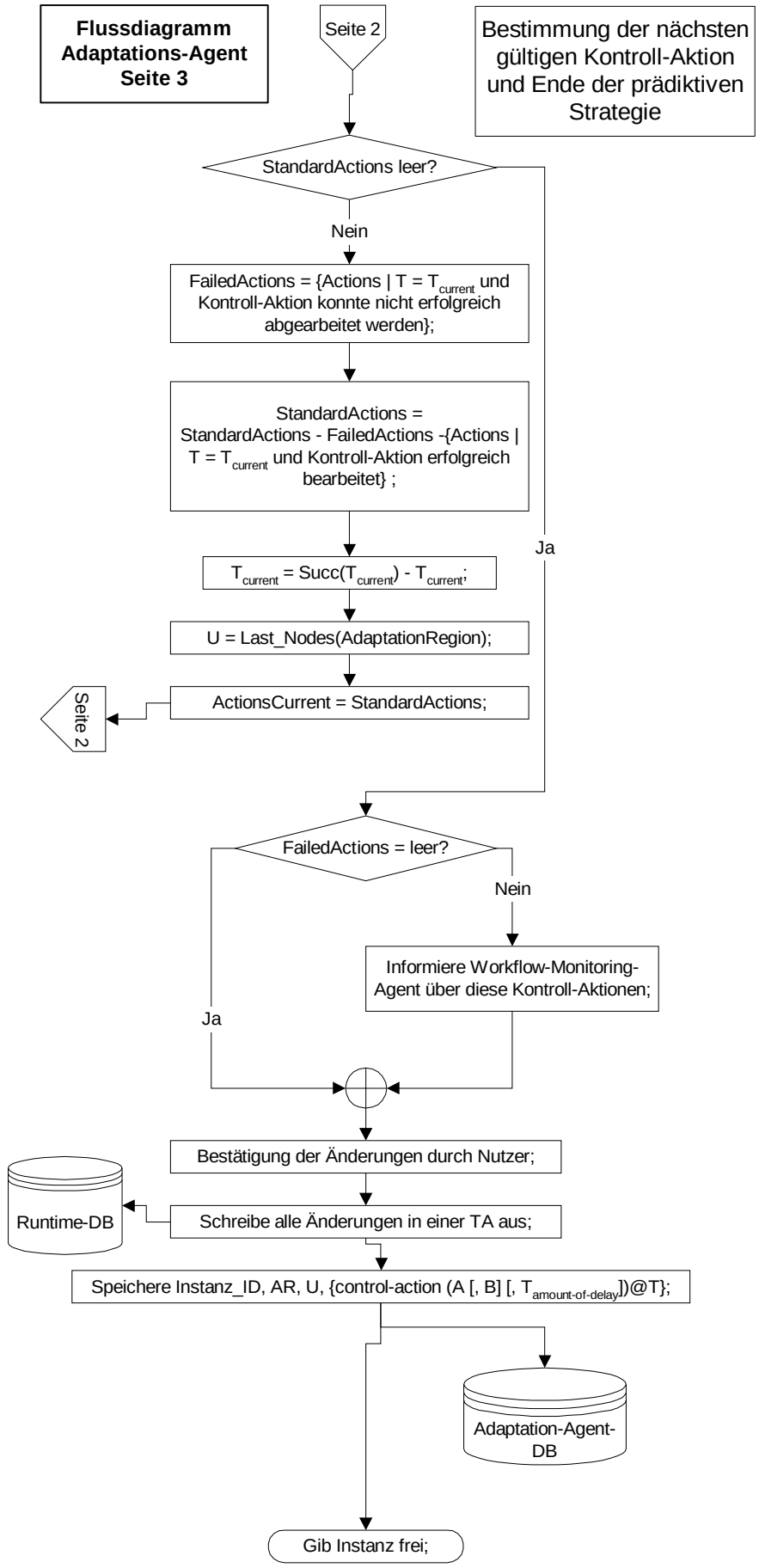


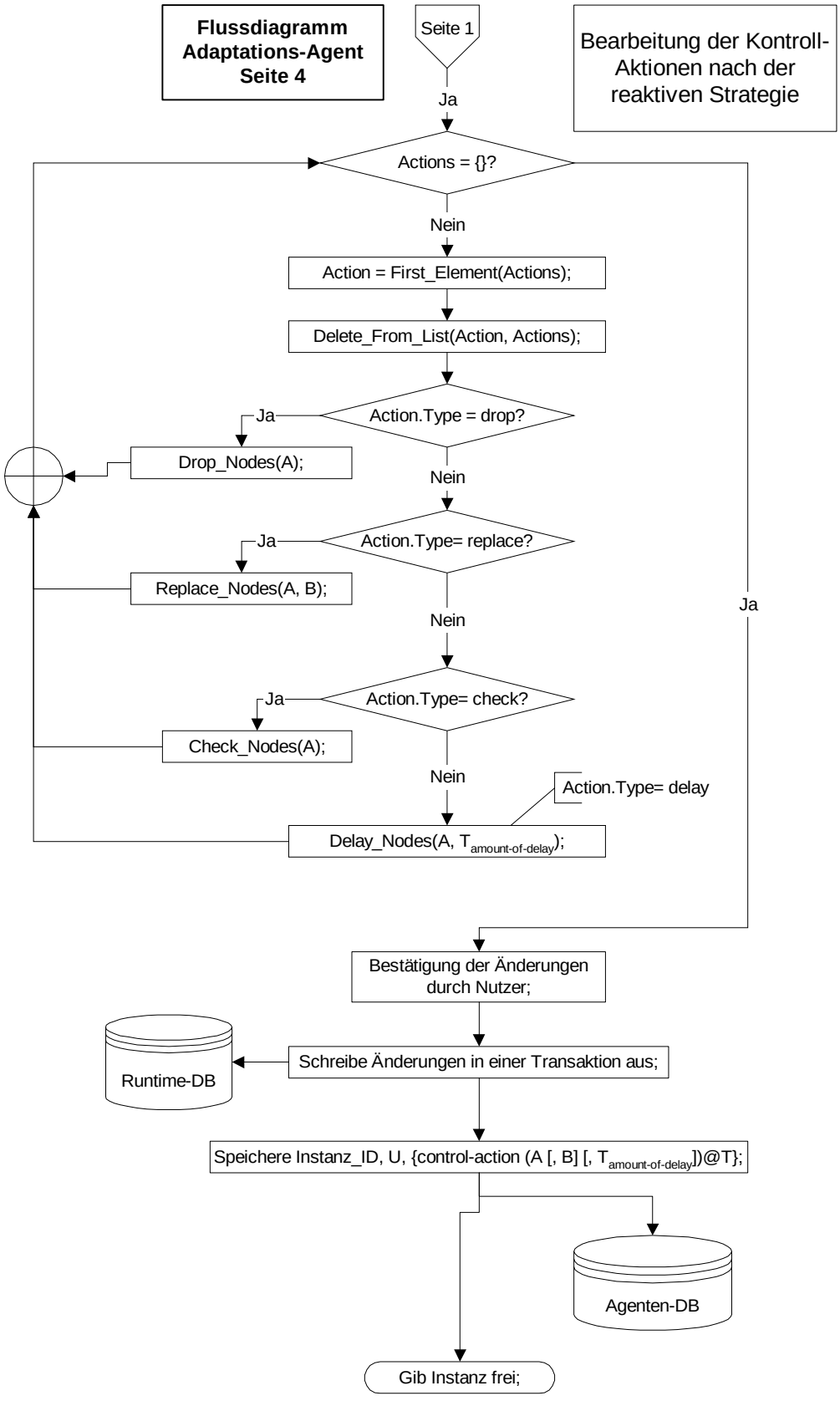
Anhang B: Flussdiagramme

B.1 Adaptations-Agent

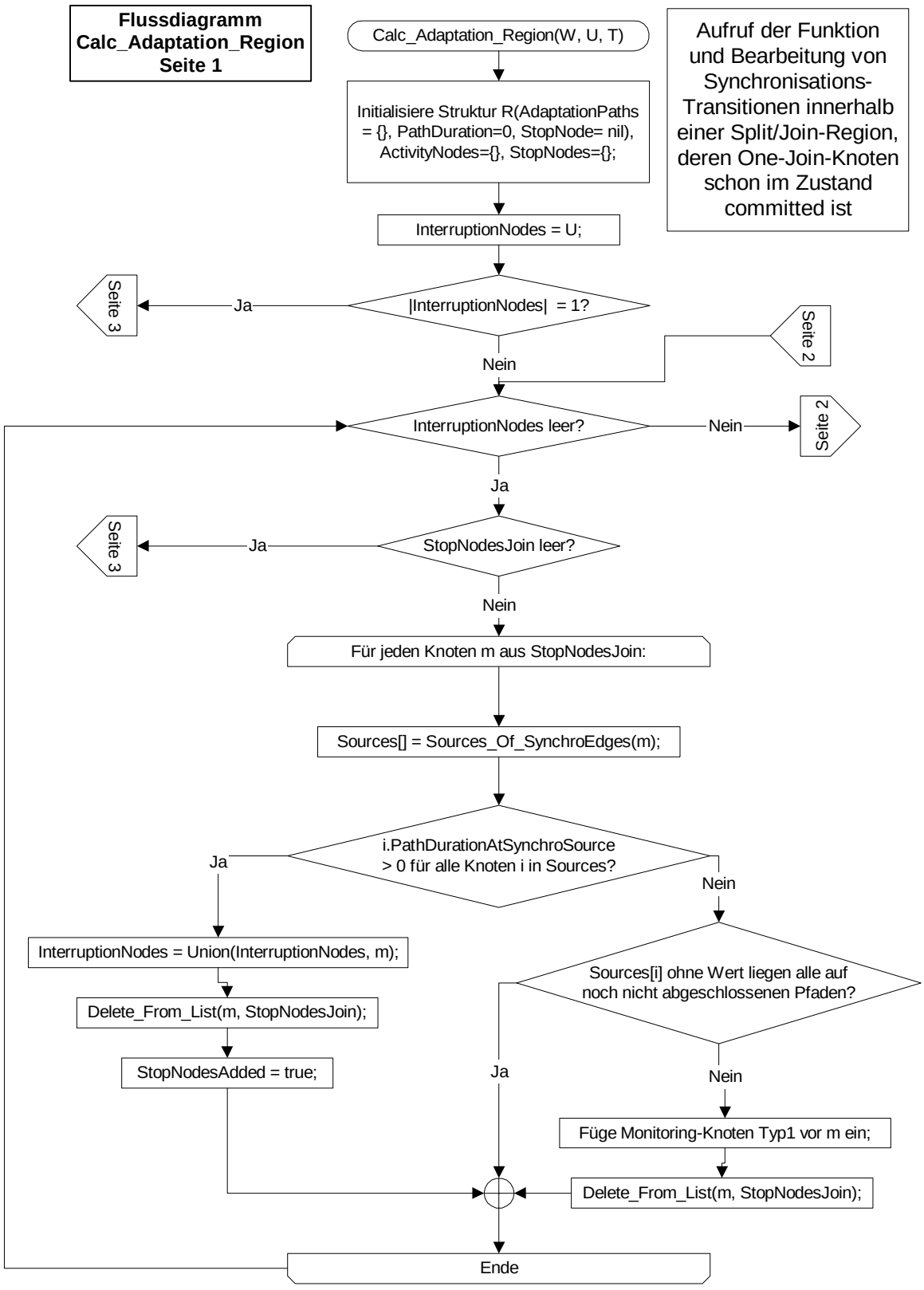


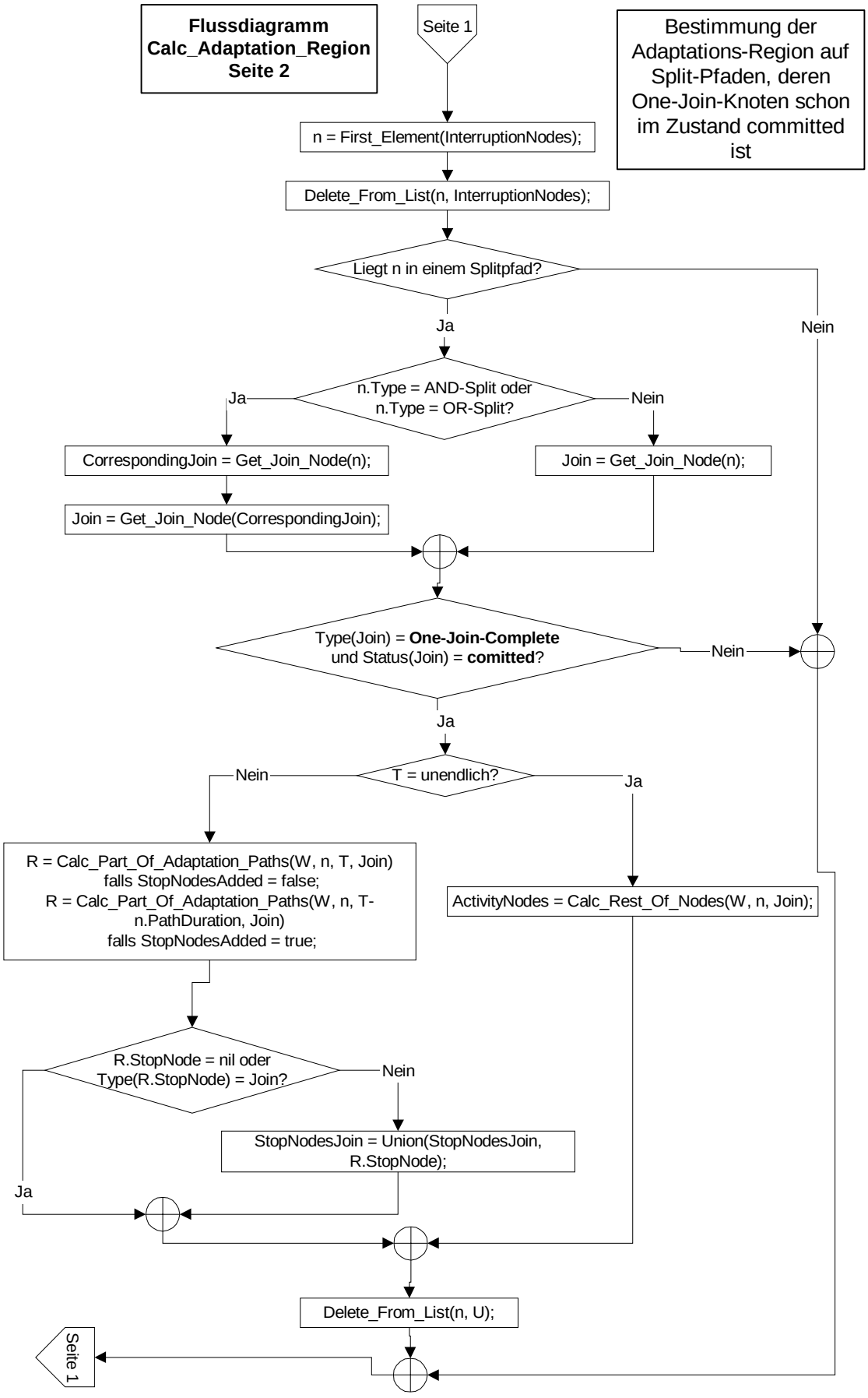

Flussdiagramm Adaptations-Agent Seite 2
Seite 1
Sortierung der Kontroll-Aktionen und Bearbeitung nach der prädiktiven Strategie
Seite 3
AR=Calc_Adaptation_Region(W, U, T current);
Speichere die zu T current gehörende AdaptationsRegion;
Für jede Kontroll-Aktion Action aus CurrentActions;
Action.Type = check?
Nein
Ja
Check_Nodes(AR, A, CurrentActions);
Delete_From_List(Action, CurrentActions);
Sortiere CurrentActions;
CurrentActions = {}?
Ja
Seite 3
Nein
Action = First_Element(CurrentActions);
Delete_From_List(Action, CurrentActions);
Ja
Action.Type = add?
Add_Node(AR perform, A);
Nein
Ja
Action.Type = drop?
Drop_Nodes(AR, A);
Nein
Ja
Action.Type = replace?
Replace_Nodes(AR, A, B);
Nein
Action.Type = delay
Delay_Nodes(AR, A, T amount-of-delay);

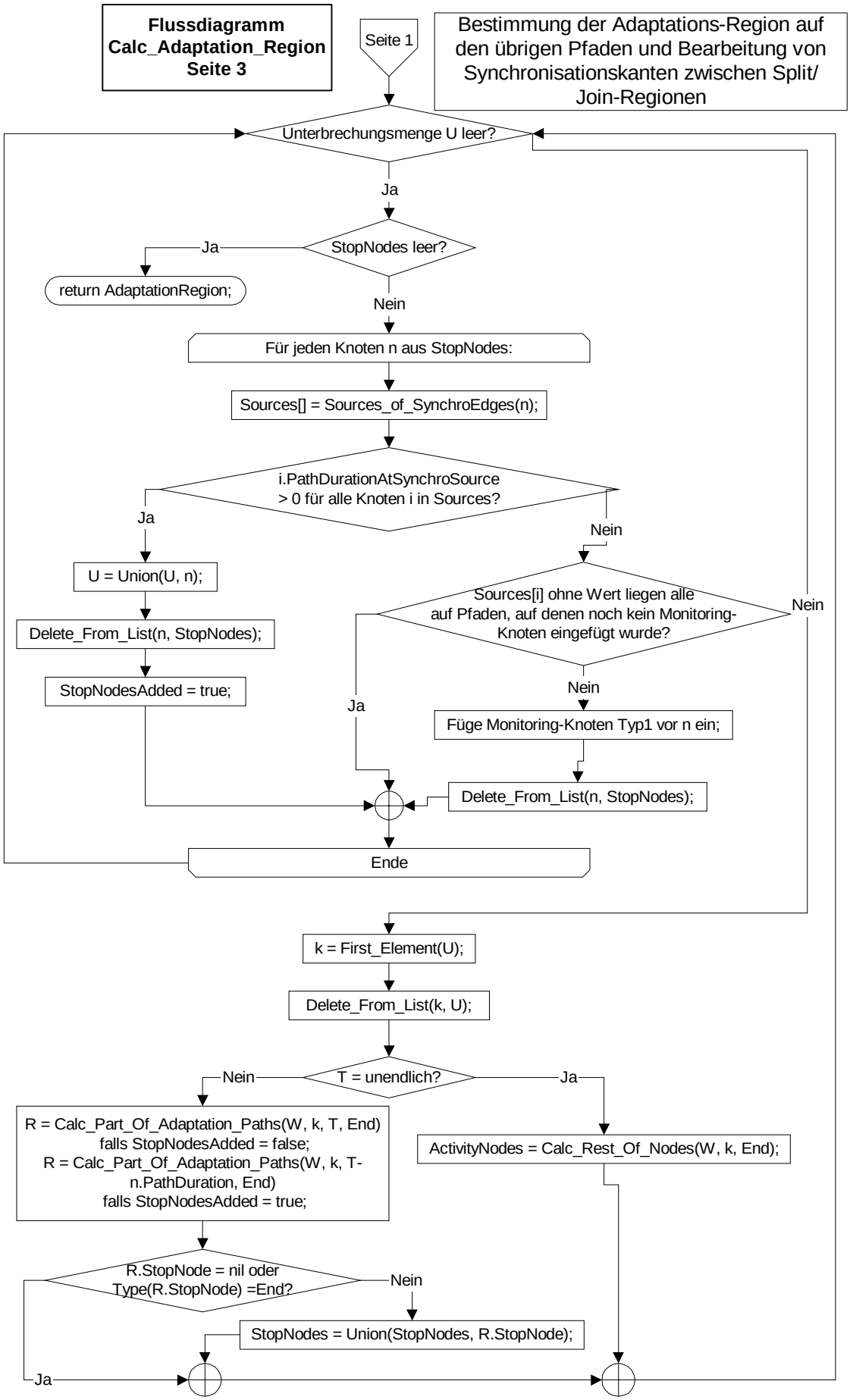

Flussdiagramm Adaptations-Agent Seite 3
Seite 2
Bestimmung der nächsten gültigen Kontroll-Aktion und Ende der prädiktiven Strategie
StandardActions leer?
Nein
Ja
FailedActions = {Actions | T = T_current und Kontroll-Aktion konnte nicht erfolgreich abgearbeitet werden};
StandardActions = StandardActions - FailedActions -{Actions | T = T_current und Kontroll-Aktion erfolgreich bearbeitet} ;
T_current = Succ(T_current) - T_current;
U = Last_Nodes(AdaptationRegion);
ActionsCurrent = StandardActions;
Seite 2
FailedActions = leer?
Nein
Ja
Informiere Workflow-Monitoring-Agent über diese Kontroll-Aktionen;
Bestätigung der Änderungen durch Nutzer;
Runtime-DB
Schreibe alle Änderungen in einer TA aus;
Speichere Instanz_ID, AR, U, {control-action (A [, B] [, T_amount-of-delay])@T};
Adaptation-Agent-DB
Gib Instanz frei;


Flussdiagramm Adaptations-Agent Seite 4
Seite 1
Bearbeitung der Kontroll-Aktionen nach der reaktiven Strategie
Ja
Actions = {}?
Nein
Action = First_Element(Actions);
Delete_From_List(Action, Actions);
Action.Type = drop?
Ja
Drop_Nodes(A);
Nein
Action.Type= replace?
Ja
Replace_Nodes(A, B);
Nein
Action.Type= check?
Ja
Check_Nodes(A);
Nein
Action.Type= delay
Delay_Nodes(A, T amount-of-delay);
Ja
Bestätigung der Änderungen durch Nutzer;
Runtime-DB
Schreibe Änderungen in einer Transaktion aus;
Speichere Instanz_ID, U, {control-action (A [, B] [, T amount-of-delay])@T};
Agenten-DB
Gib Instanz frei;

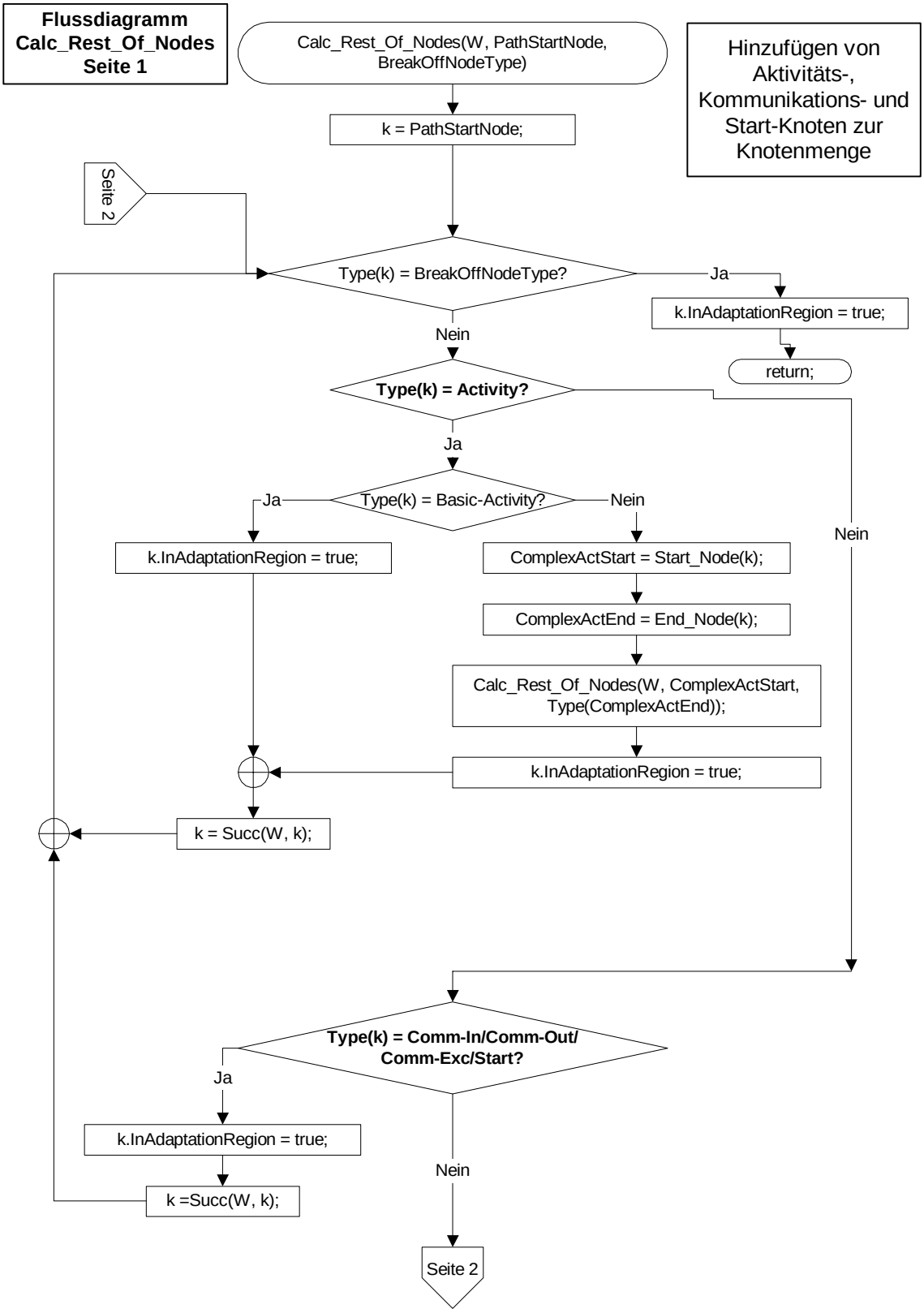
B.2 Bestimmung der Adaptations-Region

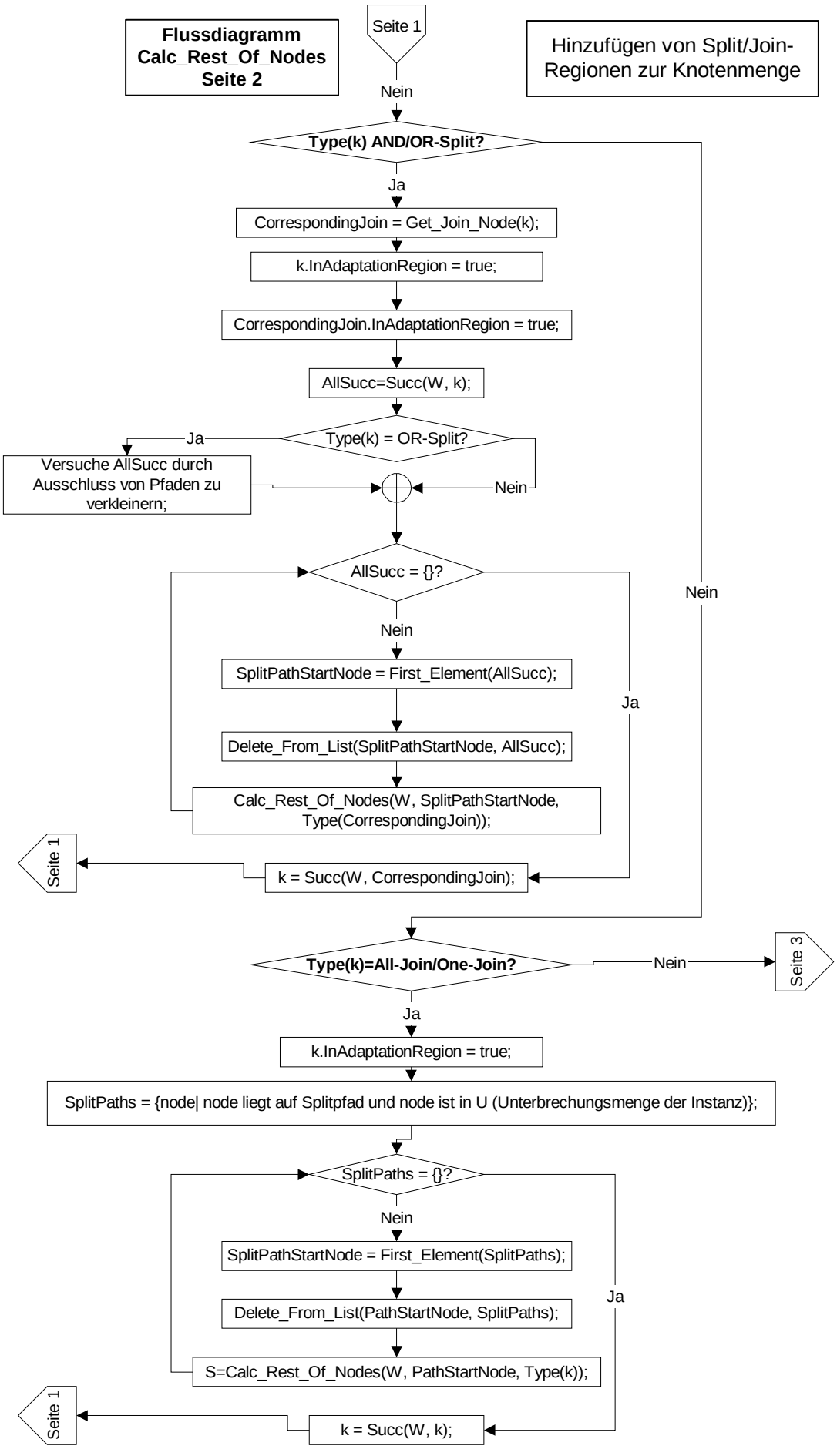


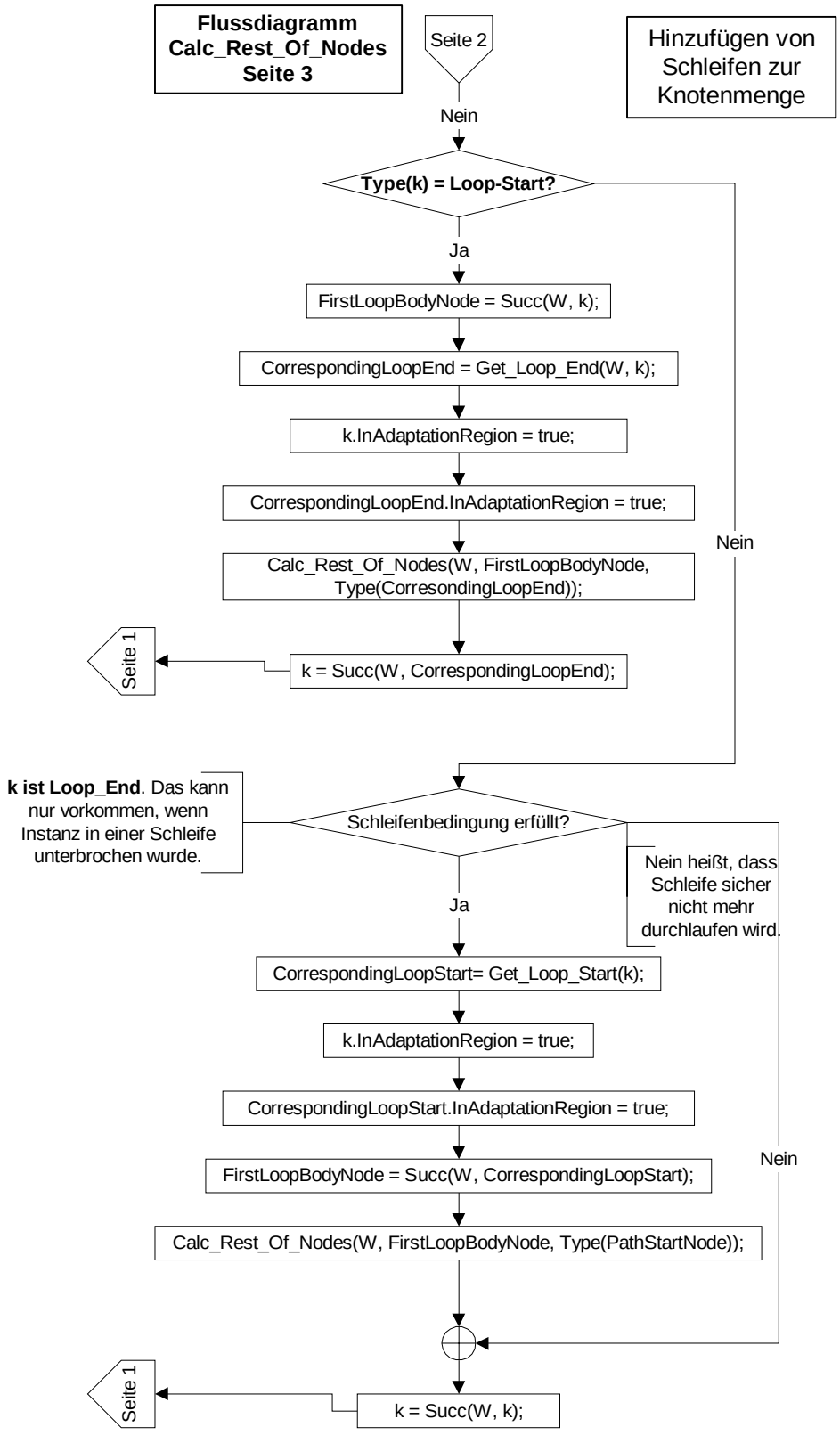

Flussdiagramm
Calc_Adaptation_Region
Seite 2
Seite 1
Bestimmung der Adaptations-Region auf Split-Pfaden, deren One-Join-Knoten schon im Zustand committed ist
n = First_Element(InterruptionNodes);
Delete_From_List(n, InterruptionNodes);
Liegt n in einem Splitpfad?
Ja
Nein
n.Type = AND-Split oder n.Type = OR-Split?
Ja
Nein
CorrespondingJoin = Get_Join_Node(n);
Join = Get_Join_Node(CorrespondingJoin);
Join = Get_Join_Node(n);
Type(Join) = One-Join-Complete und Status(Join) = comitted?
Nein
Ja
T = unendlich?
Nein
Ja
R = Calc_Part_Of_Adaptation_Paths(W, n, T, Join) falls StopNodesAdded = false; R = Calc_Part_Of_Adaptation_Paths(W, n, T-n.PathDuration, Join) falls StopNodesAdded = true;
ActivityNodes = Calc_Rest_Of_Nodes(W, n, Join);
R.StopNode = nil oder Type(R.StopNode) = Join?
Nein
StopNodesJoin = Union(StopNodesJoin, R.StopNode);
Ja
Delete_From_List(n, U);
Seite 1


Flussdiagramm
Calc_Adaptation_Region
Seite 3
Seite 1
Bestimmung der Adaptations-Region auf den übrigen Pfaden und Bearbeitung von Synchronisationskanten zwischen Split/ Join-Regionen
Unterbrechungsmenge U leer?
Ja
StopNodes leer?
Ja
return AdaptationRegion;
Nein
Für jeden Knoten n aus StopNodes:
Sources[] = Sources_of_SynchroEdges(n);
i.PathDurationAtSynchroSource > 0 für alle Knoten i in Sources?
Ja
Nein
U = Union(U, n);
Delete_From_List(n, StopNodes);
StopNodesAdded = true;
Sources[i] ohne Wert liegen alle auf Pfaden, auf denen noch kein Monitoring-Knoten eingefügt wurde?
Nein
Ja
Nein
Füge Monitoring-Knoten Typ1 vor n ein;
Delete_From_List(n, StopNodes);
Ende
k = First_Element(U);
Delete_From_List(k, U);
T = unendlich?
Nein
Ja
R = Calc_Part_Of_Adaptation_Paths(W, k, T, End) falls StopNodesAdded = false;
R = Calc_Part_Of_Adaptation_Paths(W, k, T-n.PathDuration, End) falls StopNodesAdded = true;
ActivityNodes = Calc_Rest_Of_Nodes(W, k, End);
R.StopNode = nil oder Type(R.StopNode) =End?
Nein
StopNodes = Union(StopNodes, R.StopNode);
Ja

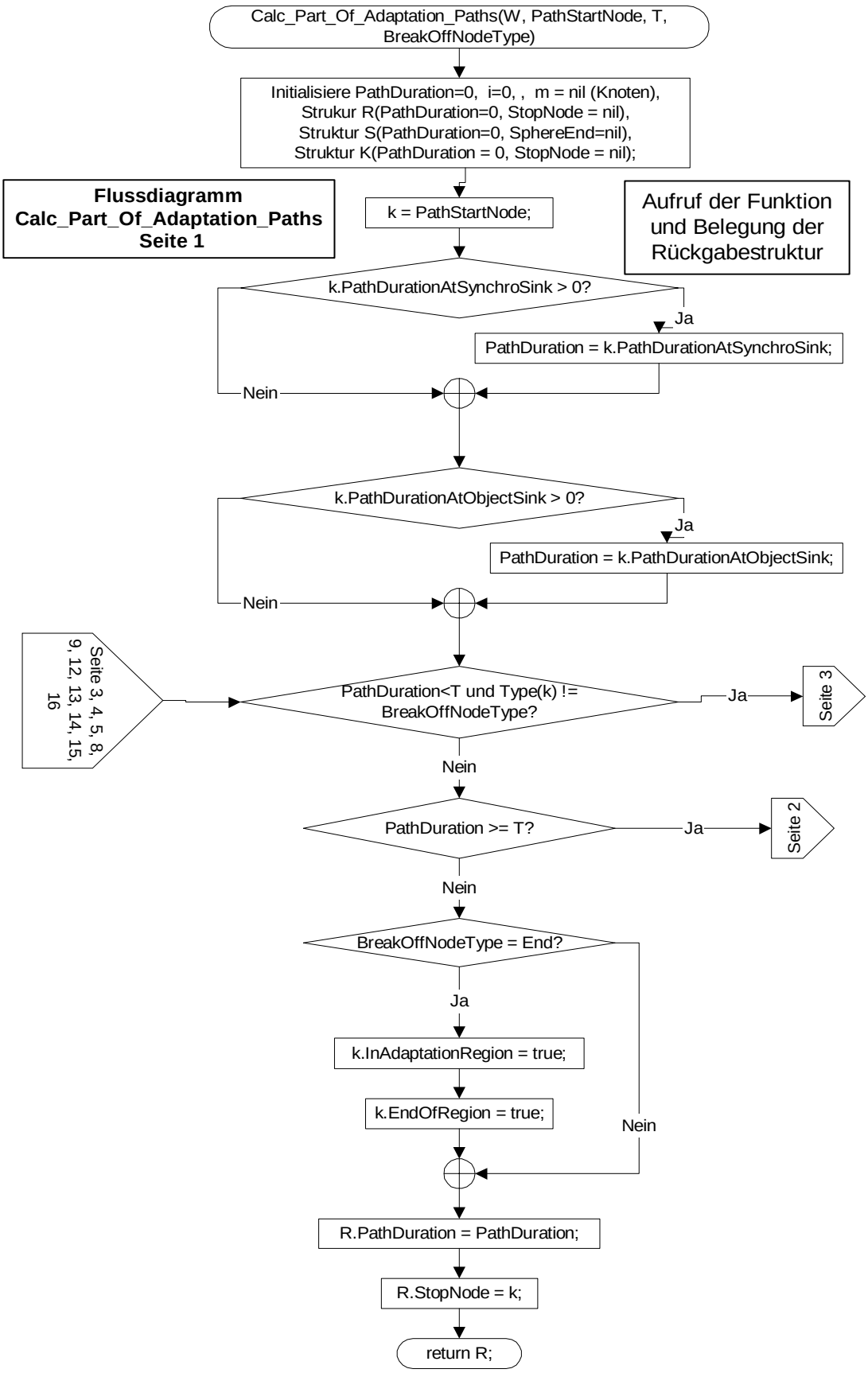
B3. Bestimmung der Knotenmenge der Adaptations-Region für T = unendlich

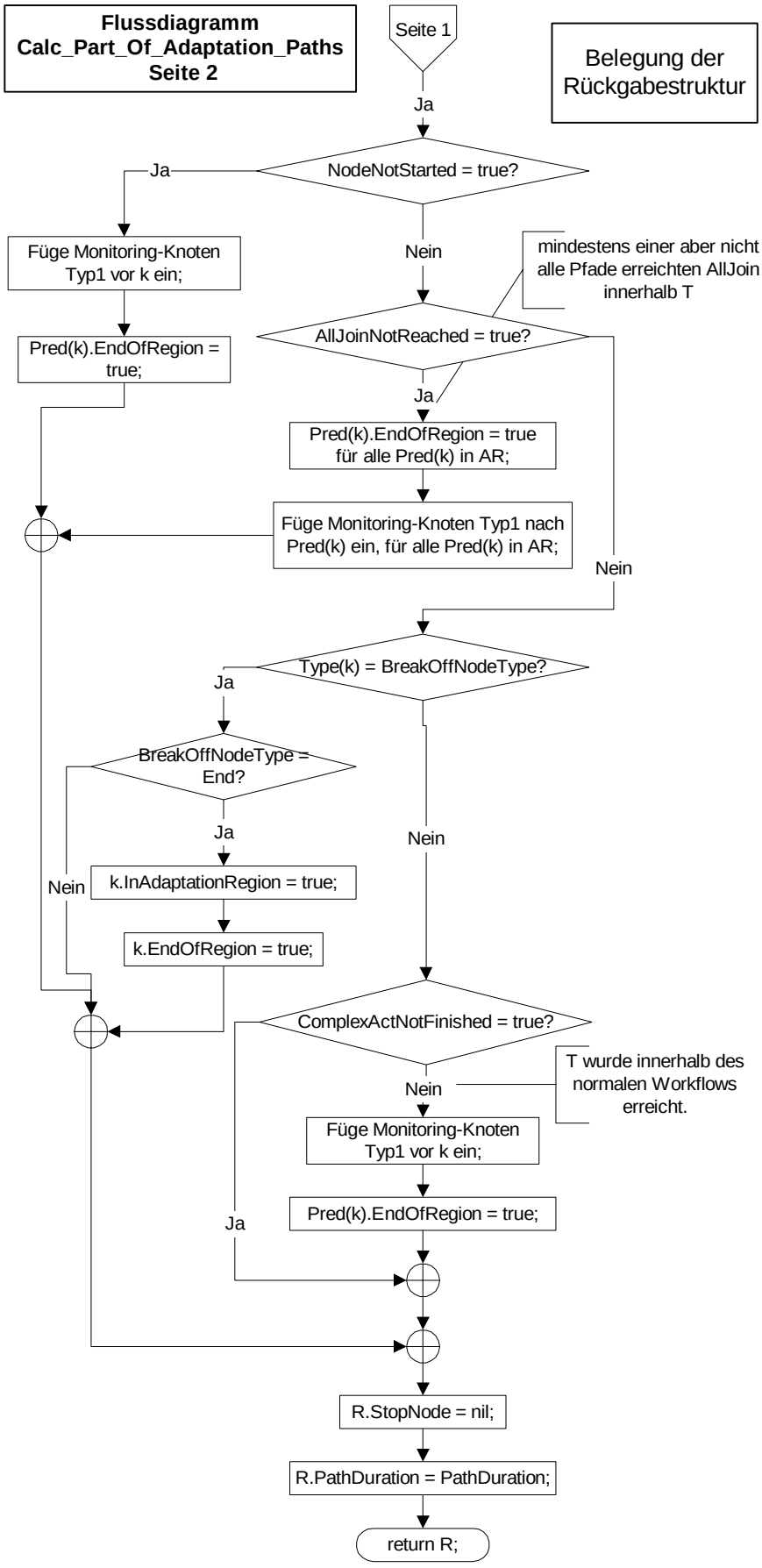


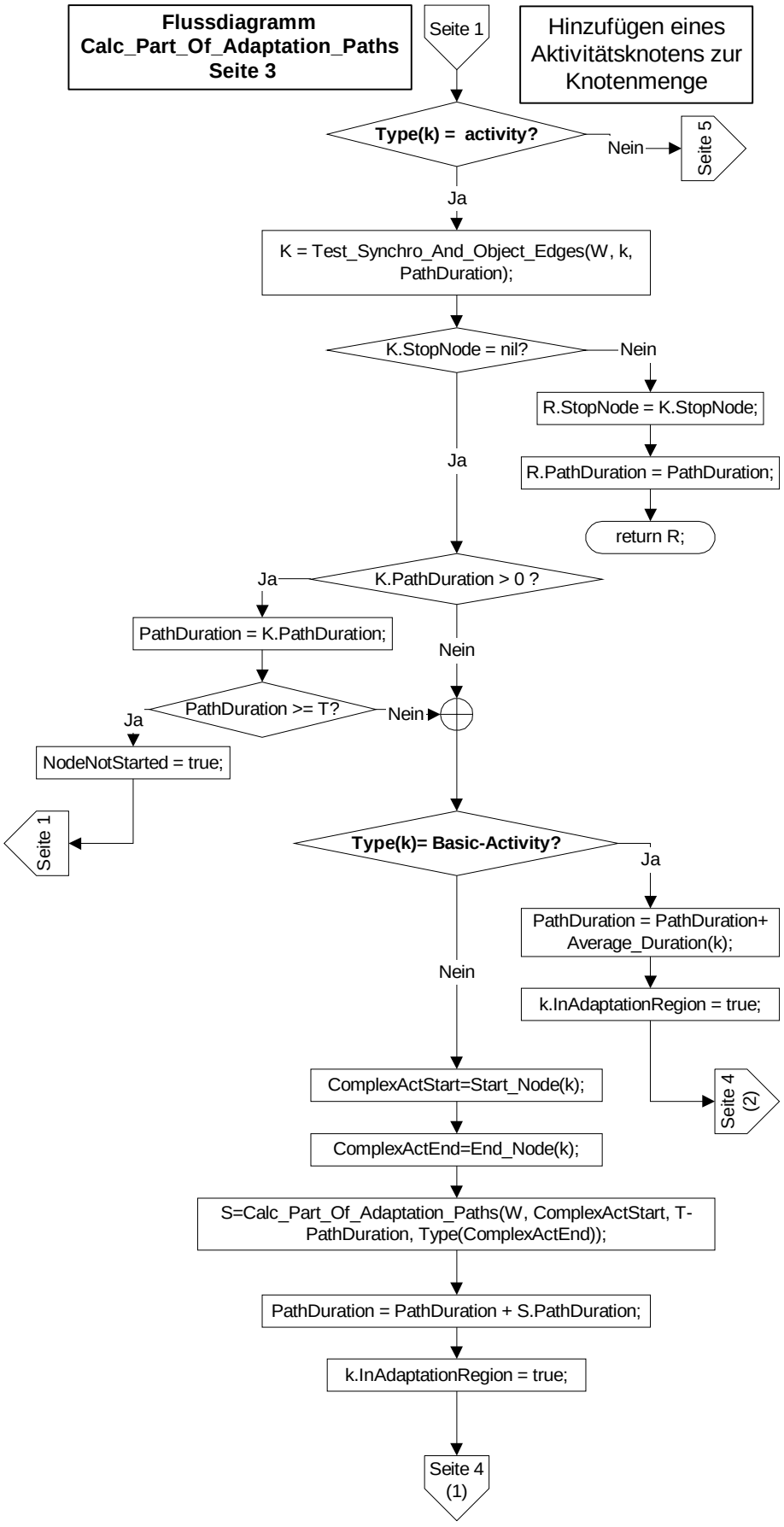

Flussdiagramm Calc_Rest_Of_Nodes Seite 2
Seite 1
Hinzufügen von Split/Join-Regionen zur Knotenmenge
Nein
Type(k) AND/OR-Split?
Ja
CorrespondingJoin = Get_Join_Node(k);
k.InAdaptationRegion = true;
CorrespondingJoin.InAdaptationRegion = true;
AllSucc=Succ(W, k);
Type(k) = OR-Split?
Ja
Versuche AllSucc durch Ausschluss von Pfaden zu verkleinern;
Nein
AllSucc = {}?
Nein
SplitPathStartNode = First_Element(AllSucc);
Delete_From_List(SplitPathStartNode, AllSucc);
Calc_Rest_Of_Nodes(W, SplitPathStartNode, Type(CorrespondingJoin));
Ja
Seite 1
k = Succ(W, CorrespondingJoin);
Nein
Type(k)=All-Join/One-Join?
Nein
Seite 3
Ja
k.InAdaptationRegion = true;
SplitPaths = {node| node liegt auf Splitpfad und node ist in U (Unterbrechungsmenge der Instanz)};
SplitPaths = {}?
Nein
SplitPathStartNode = First_Element(SplitPaths);
Delete_From_List(PathStartNode, SplitPaths);
S=Calc_Rest_Of_Nodes(W, PathStartNode, Type(k));
Ja
Seite 1
k = Succ(W, k);

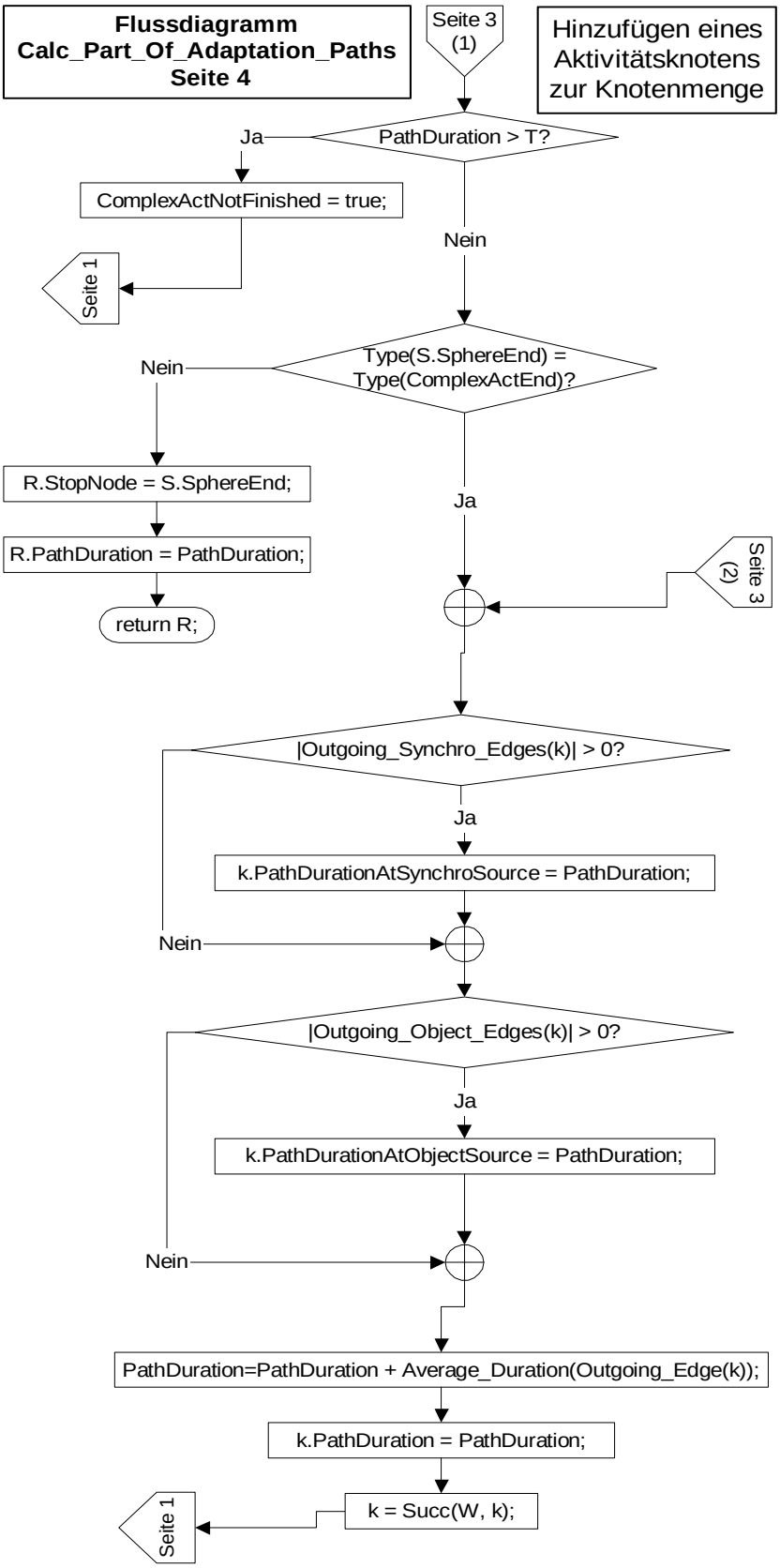

Flussdiagramm
Calc_Rest_Of_Nodes
Seite 3
Seite 2
Hinzufügen von Schleifen zur Knotenmenge
Nein
Type(k) = Loop-Start?
Ja
FirstLoopBodyNode = Succ(W, k);
CorrespondingLoopEnd = Get_Loop_End(W, k);
k.InAdaptationRegion = true;
CorrespondingLoopEnd.InAdaptationRegion = true;
Nein
Calc_Rest_Of_Nodes(W, FirstLoopBodyNode, Type(CorresondingLoopEnd));
Seite 1
k = Succ(W, CorrespondingLoopEnd);
k ist Loop_End. Das kann nur vorkommen, wenn Instanz in einer Schleife unterbrochen wurde.
Schleifenbedingung erfüllt?
Nein heißt, dass Schleife sicher nicht mehr durchlaufen wird.
Ja
CorrespondingLoopStart= Get_Loop_Start(k);
k.InAdaptationRegion = true;
CorrespondingLoopStart.InAdaptationRegion = true;
Nein
FirstLoopBodyNode = Succ(W, CorrespondingLoopStart);
Calc_Rest_Of_Nodes(W, FirstLoopBodyNode, Type(PathStartNode));
Seite 1
k = Succ(W, k);

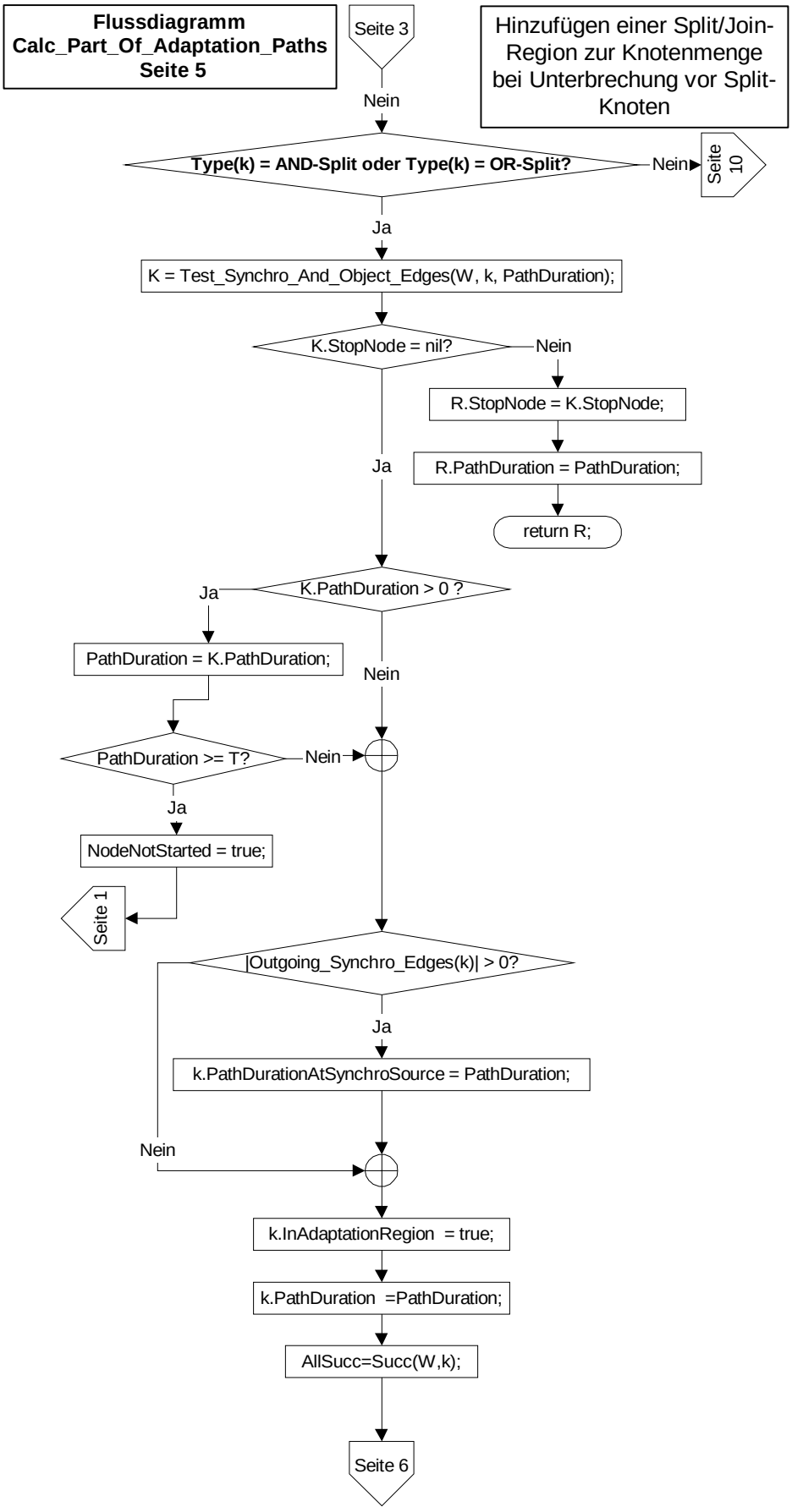
B4. Bestimmung der Knotenmenge der Adaptations-Region für T < unendlich

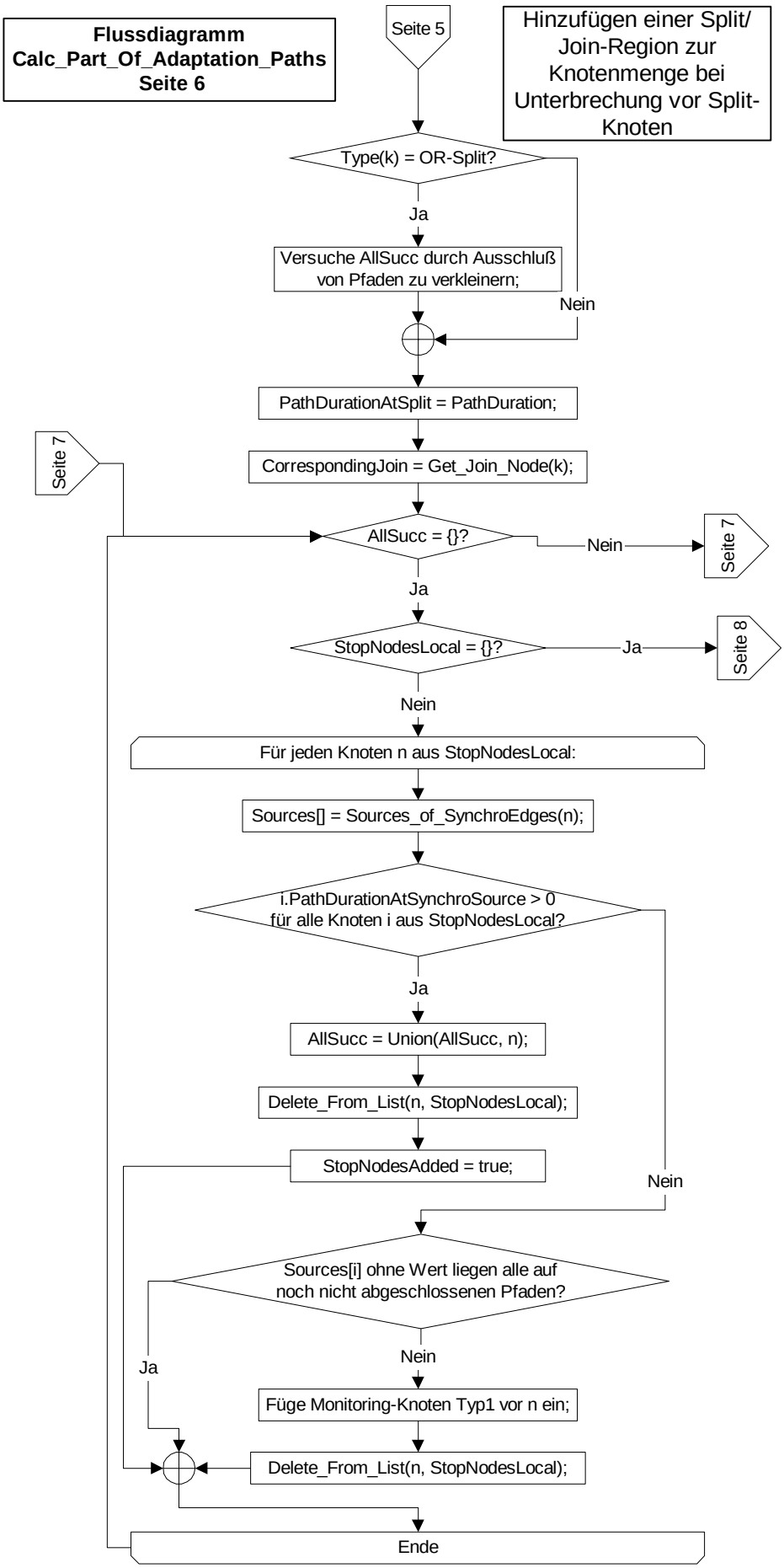


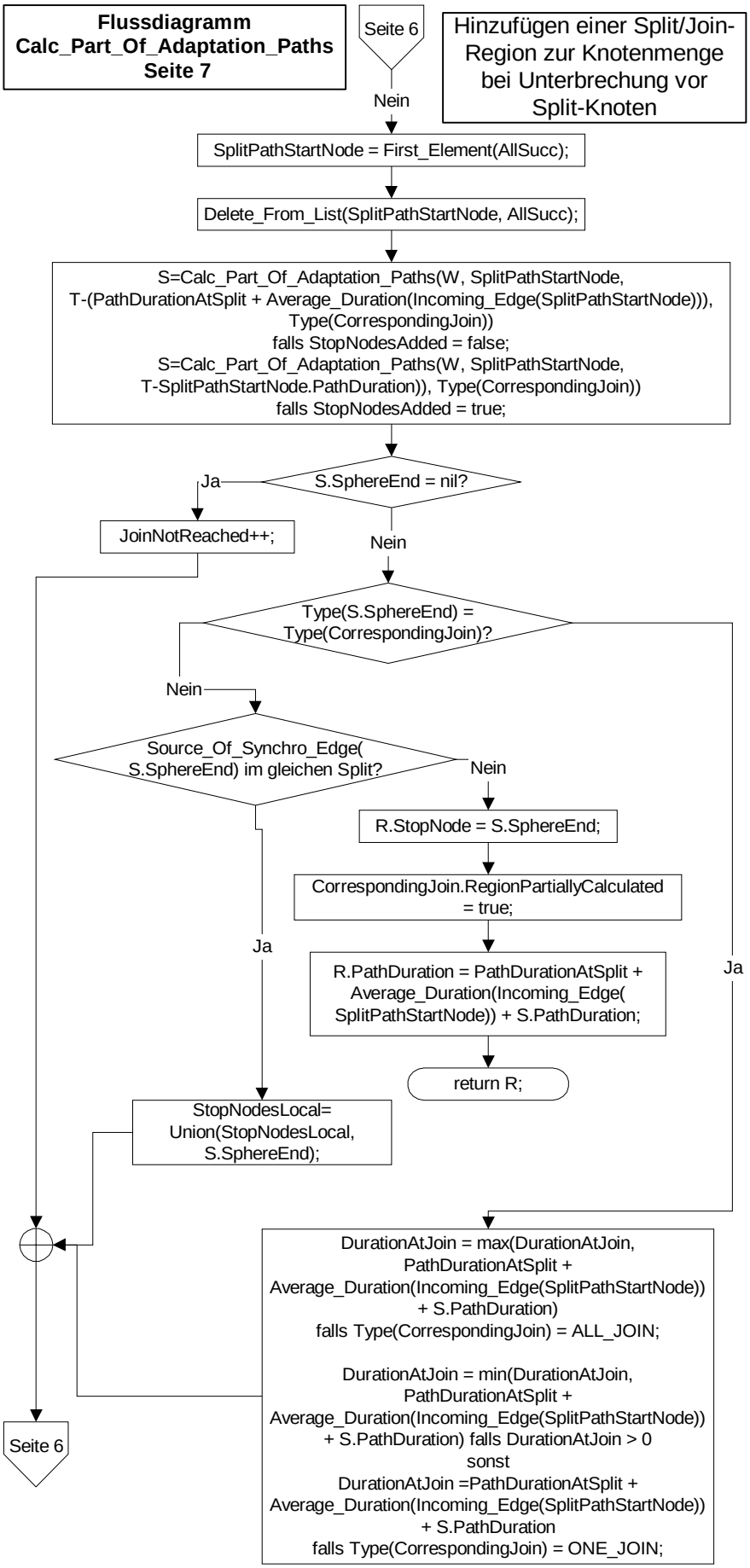

Flussdiagramm
Calc_Part_Of_Adaptation_Paths
Seite 2
Seite 1
Belegung der Rückgabestruktur
Ja
NodeNotStarted = true?
Ja
Nein
Füge Monitoring-Knoten Typ1 vor k ein;
mindestens einer aber nicht alle Pfade erreichten AllJoin innerhalb T
Pred(k).EndOfRegion = true;
AllJoinNotReached = true?
Ja
Pred(k).EndOfRegion = true für alle Pred(k) in AR;
Füge Monitoring-Knoten Typ1 nach Pred(k) ein, für alle Pred(k) in AR;
Nein
Type(k) = BreakOffNodeType?
Ja
BreakOffNodeType = End?
Ja
Nein
Nein
k.InAdaptationRegion = true;
k.EndOfRegion = true;
ComplexActNotFinished = true?
T wurde innerhalb des normalen Workflows erreicht.
Nein
Füge Monitoring-Knoten Typ1 vor k ein;
Pred(k).EndOfRegion = true;
Ja
R.StopNode = nil;
R.PathDuration = PathDuration;
return R;

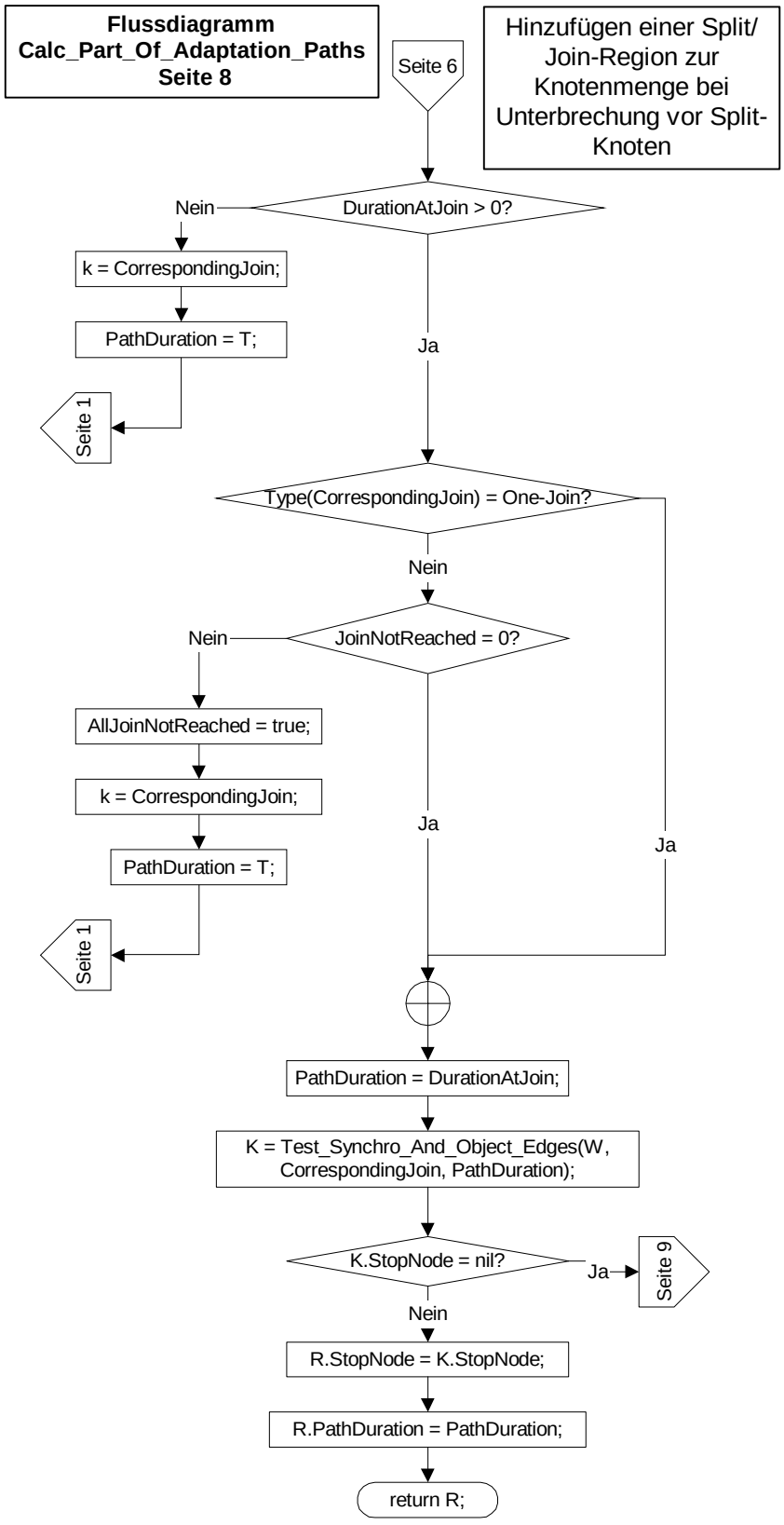

Flussdiagramm
Calc_Part_Of_Adaptation_Paths
Seite 3
Seite 1
Hinzufügen eines Aktivitätsknotens zur Knotenmenge
Type(k) = activity?
Nein
Seite 5
Ja
K = Test_Synchro_And_Object_Edges(W, k, PathDuration);
K.StopNode = nil?
Nein
R.StopNode = K.StopNode;
R.PathDuration = PathDuration;
return R;
Ja
Ja
K.PathDuration > 0 ?
PathDuration = K.PathDuration;
Nein
PathDuration >= T?
Nein
Ja
NodeNotStarted = true;
Seite 1
Type(k)= Basic-Activity?
Ja
PathDuration = PathDuration+ Average_Duration(k);
Nein
k.InAdaptationRegion = true;
ComplexActStart=Start_Node(k);
Seite 4 (2)
ComplexActEnd=End_Node(k);
S=Calc_Part_Of_Adaptation_Paths(W, ComplexActStart, T-PathDuration, Type(ComplexActEnd));
PathDuration = PathDuration + S.PathDuration;
k.InAdaptationRegion = true;
Seite 4 (1)

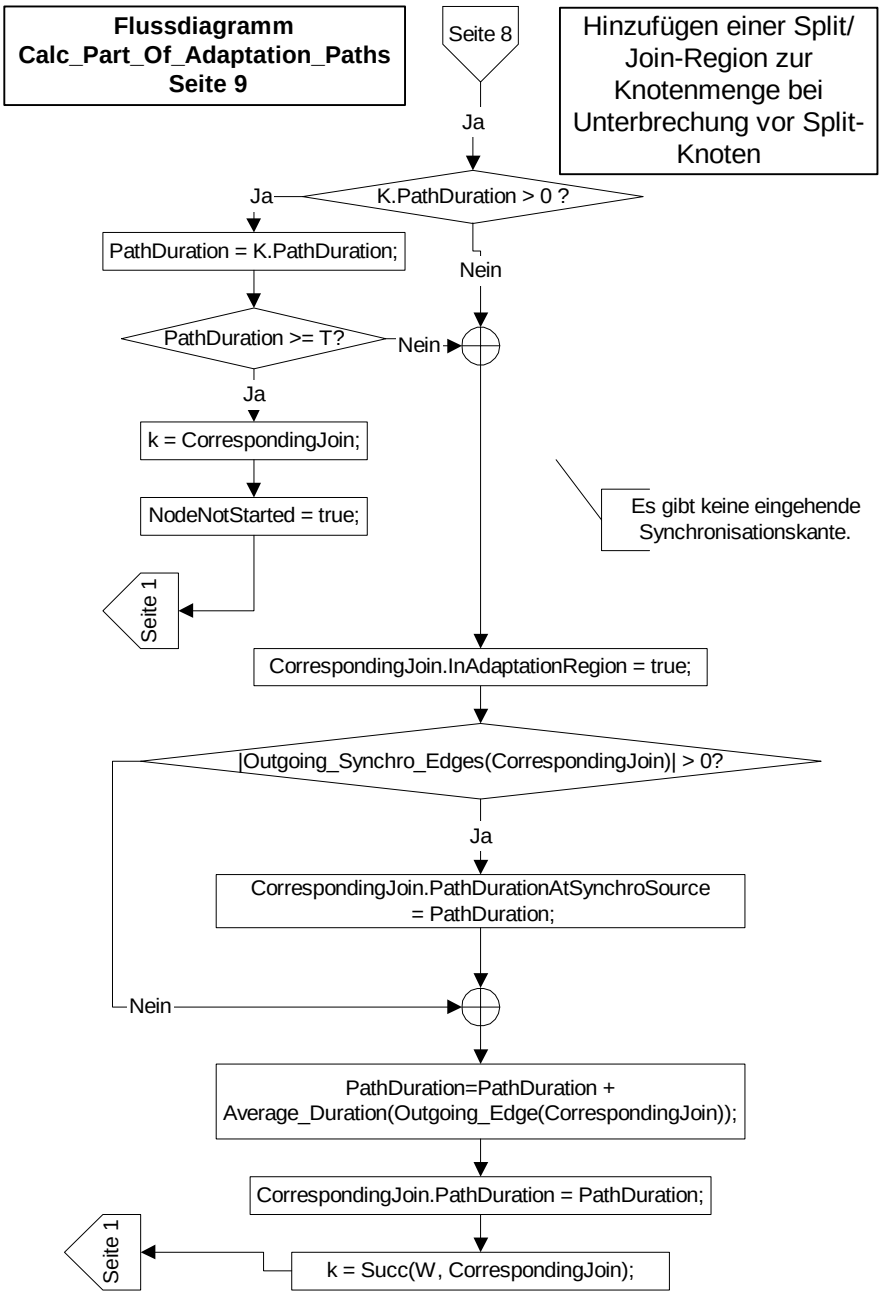

Flussdiagramm
Calc_Part_Of_Adaptation_Paths
Seite 4
Seite 3
(1)
Hinzufügen eines
Aktivitätsknotens
zur Knotenmenge
PathDuration > T?
Ja
ComplexActNotFinished = true;
Seite 1
Nein
Type(S.SphereEnd) =
Type(ComplexActEnd)?
Nein
R.StopNode = S.SphereEnd;
R.PathDuration = PathDuration;
return R;
Ja
Seite 3
(2)
|Outgoing_Synchro_Edges(k)| > 0?
Ja
k.PathDurationAtSynchroSource = PathDuration;
Nein
|Outgoing_Object_Edges(k)| > 0?
Ja
k.PathDurationAtObjectSource = PathDuration;
Nein
PathDuration=PathDuration + Average_Duration(Outgoing_Edge(k));
k.PathDuration = PathDuration;
k = Succ(W, k);
Seite 1

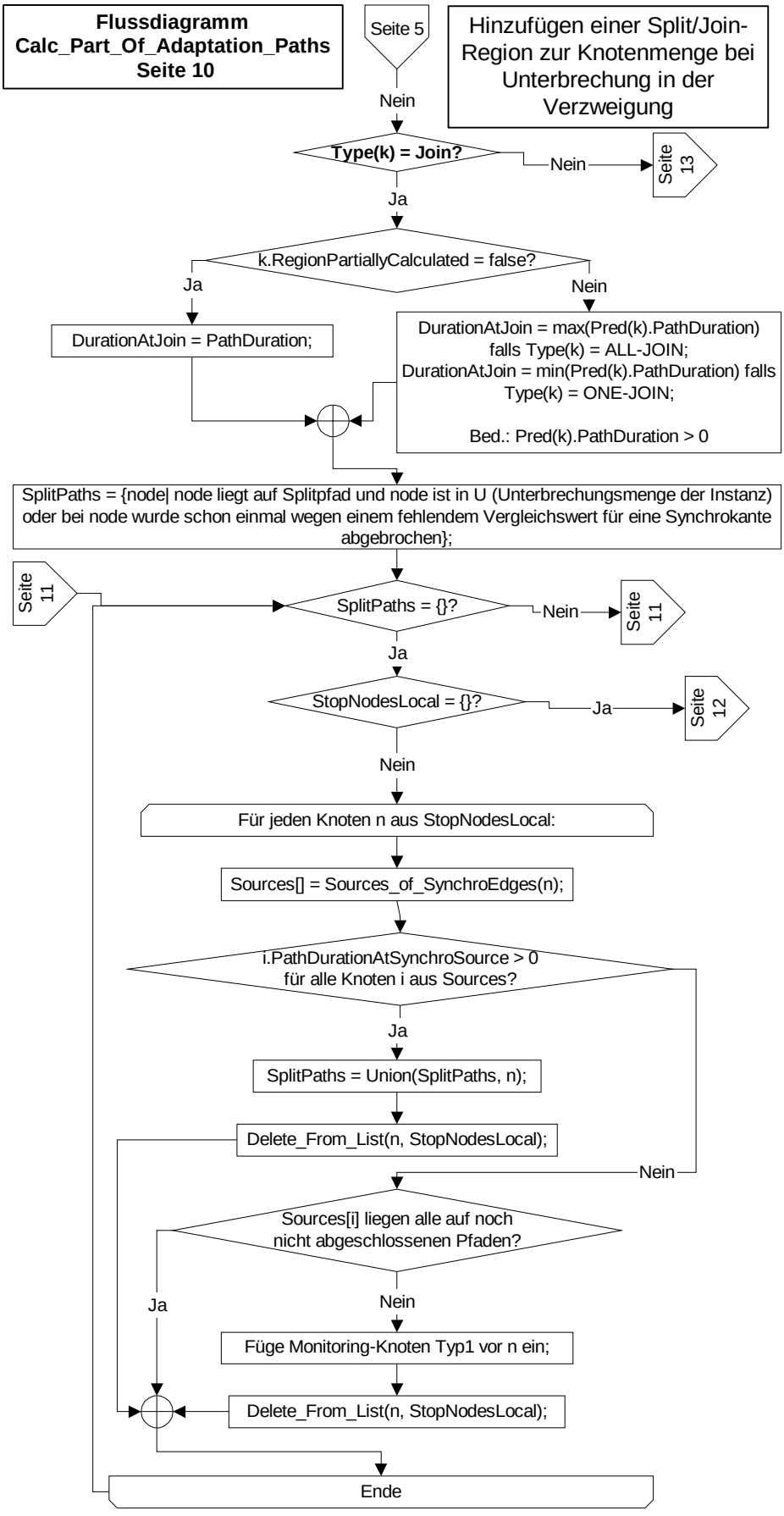

Flussdiagramm
Calc_Part_Of_Adaptation_Paths
Seite 5
Seite 3
Hinzufügen einer Split/Join-Region zur Knotenmenge bei Unterbrechung vor Split-Knoten
Nein
Type(k) = AND-Split oder Type(k) = OR-Split?
Nein
Seite 10
Ja
K = Test_Synchro_And_Object_Edges(W, k, PathDuration);
K.StopNode = nil?
Nein
R.StopNode = K.StopNode;
R.PathDuration = PathDuration;
return R;
Ja
K.PathDuration > 0 ?
Ja
PathDuration = K.PathDuration;
Nein
PathDuration >= T?
Nein
Ja
NodeNotStarted = true;
Seite 1
|Outgoing_Synchro_Edges(k)| > 0?
Ja
k.PathDurationAtSynchroSource = PathDuration;
Nein
k.InAdaptationRegion = true;
k.PathDuration =PathDuration;
AllSucc=Succ(W,k);
Seite 6

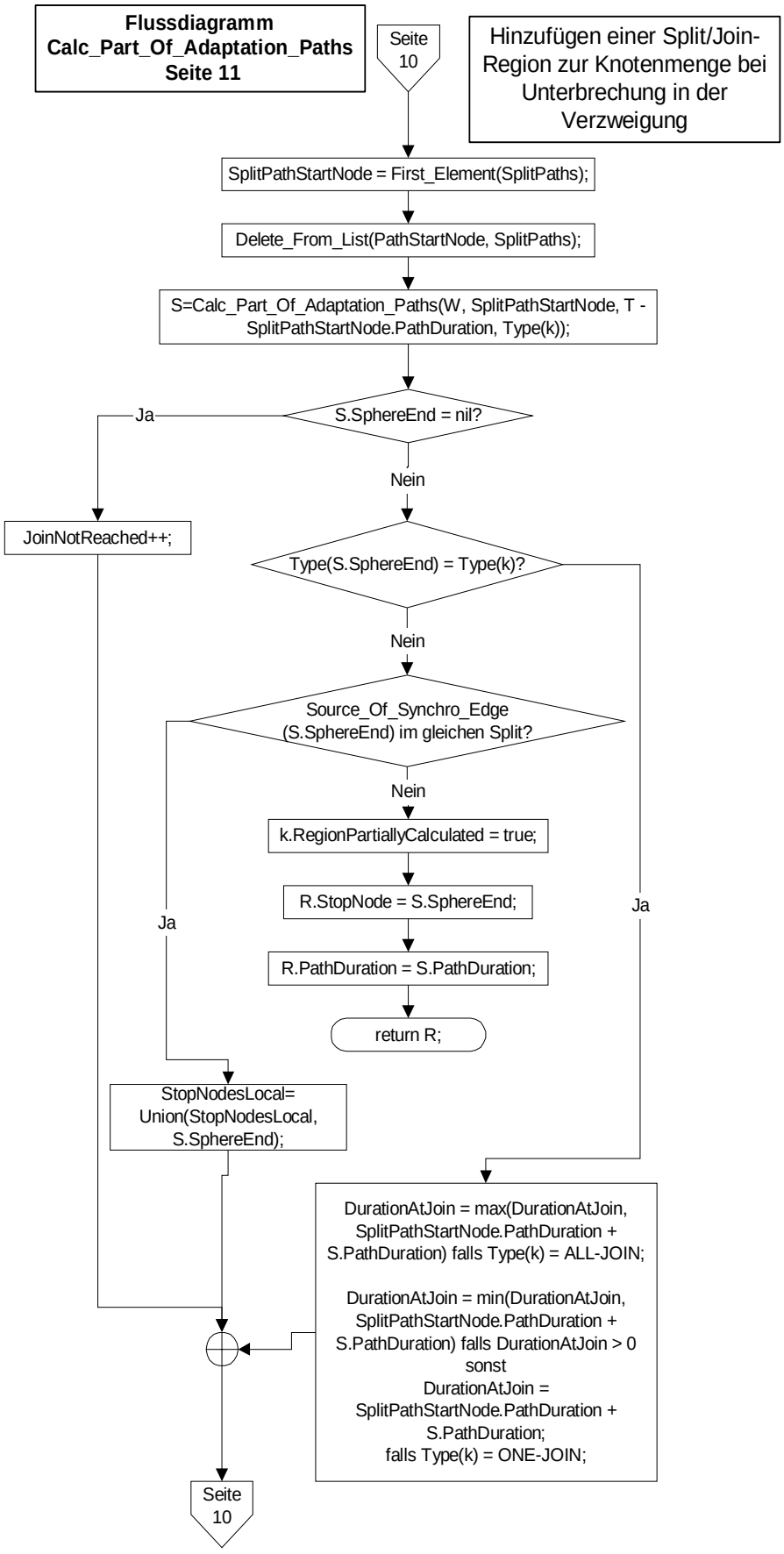

Flussdiagramm
Calc_Part_Of_Adaptation_Paths
Seite 6
Seite 5
Hinzufügen einer Split/ Join-Region zur Knotenmenge bei Unterbrechung vor Split-Knoten
Type(k) = OR-Split?
Ja
Versuche AllSucc durch Ausschluß von Pfaden zu verkleinern;
Nein
PathDurationAtSplit = PathDuration;
Seite 7
CorrespondingJoin = Get_Join_Node(k);
AllSucc = {}?
Nein
Seite 7
Ja
StopNodesLocal = {}?
Ja
Seite 8
Nein
Für jeden Knoten n aus StopNodesLocal:
Sources[] = Sources_of_SynchroEdges(n);
i.PathDurationAtSynchroSource > 0
für alle Knoten i aus StopNodesLocal?
Ja
AllSucc = Union(AllSucc, n);
Delete_From_List(n, StopNodesLocal);
StopNodesAdded = true;
Nein
Sources[i] ohne Wert liegen alle auf
noch nicht abgeschlossenen Pfaden?
Nein
Ja
Füge Monitoring-Knoten Typ1 vor n ein;
Delete_From_List(n, StopNodesLocal);
Ende

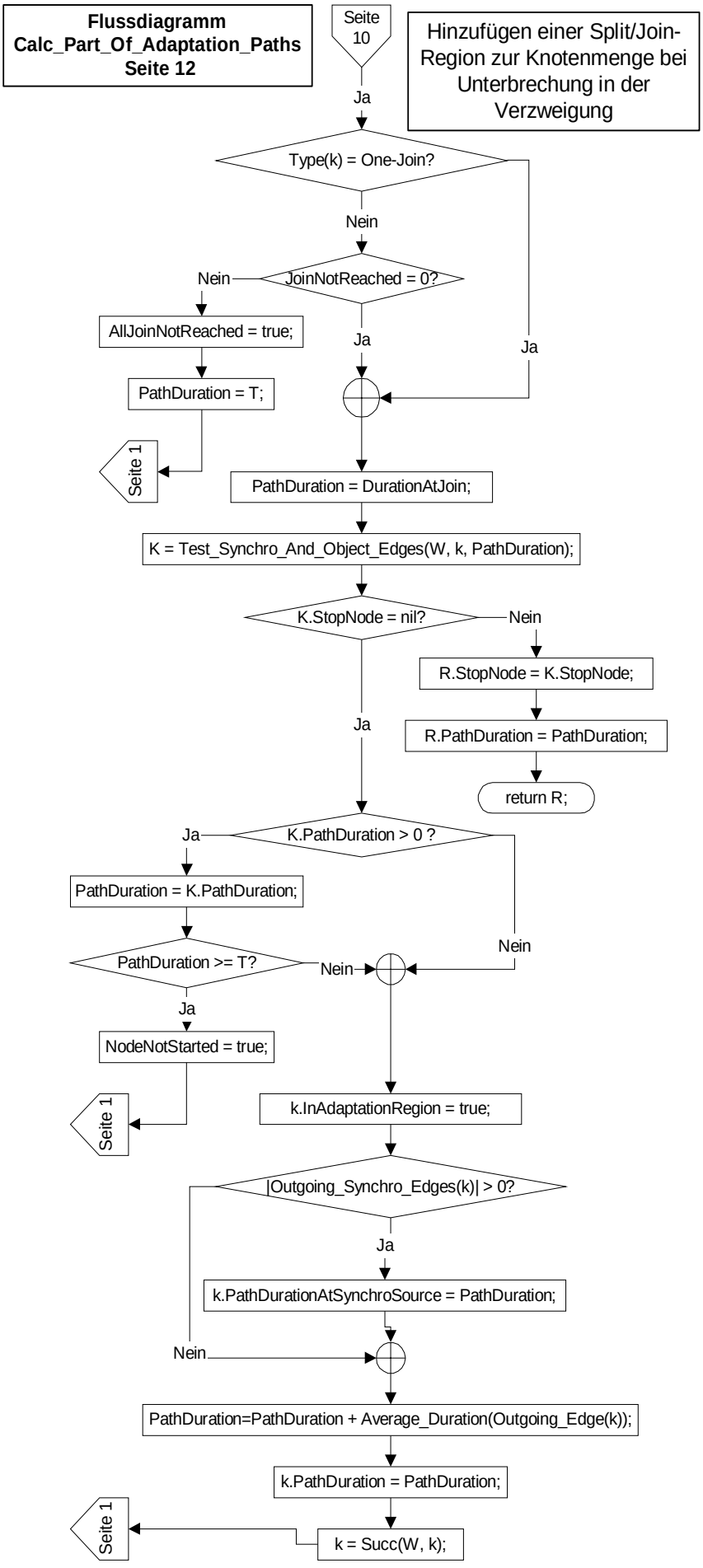

Flussdiagramm
Calc_Part_Of_Adaptation_Paths
Seite 7
Seite 6
Hinzufügen einer Split/Join-Region zur Knotenmenge bei Unterbrechung vor Split-Knoten
Nein
SplitPathStartNode = First_Element(AllSucc);
Delete_From_List(SplitPathStartNode, AllSucc);
S=Calc_Part_Of_Adaptation_Paths(W, SplitPathStartNode,
T-(PathDurationAtSplit + Average_Duration(Incoming_Edge(SplitPathStartNode))),
Type(CorrespondingJoin))
falls StopNodesAdded = false;
S=Calc_Part_Of_Adaptation_Paths(W, SplitPathStartNode,
T-SplitPathStartNode.PathDuration)), Type(CorrespondingJoin))
falls StopNodesAdded = true;
S.SphereEnd = nil?
Ja
JoinNotReached++;
Nein
Type(S.SphereEnd) =
Type(CorrespondingJoin)?
Nein
Source_Of_Synchro_Edge(
S.SphereEnd) im gleichen Split?
Nein
R.StopNode = S.SphereEnd;
CorrespondingJoin.RegionPartiallyCalculated
= true;
Ja
R.PathDuration = PathDurationAtSplit +
Average_Duration(Incoming_Edge(
SplitPathStartNode)) + S.PathDuration;
Ja
return R;
StopNodesLocal=
Union(StopNodesLocal,
S.SphereEnd);
DurationAtJoin = max(DurationAtJoin,
PathDurationAtSplit +
Average_Duration(Incoming_Edge(SplitPathStartNode))
+ S.PathDuration)
falls Type(CorrespondingJoin) = ALL_JOIN;
DurationAtJoin = min(DurationAtJoin,
PathDurationAtSplit +
Average_Duration(Incoming_Edge(SplitPathStartNode))
+ S.PathDuration) falls DurationAtJoin > 0
sonst
DurationAtJoin =PathDurationAtSplit +
Average_Duration(Incoming_Edge(SplitPathStartNode))
+ S.PathDuration
falls Type(CorrespondingJoin) = ONE_JOIN;
Seite 6

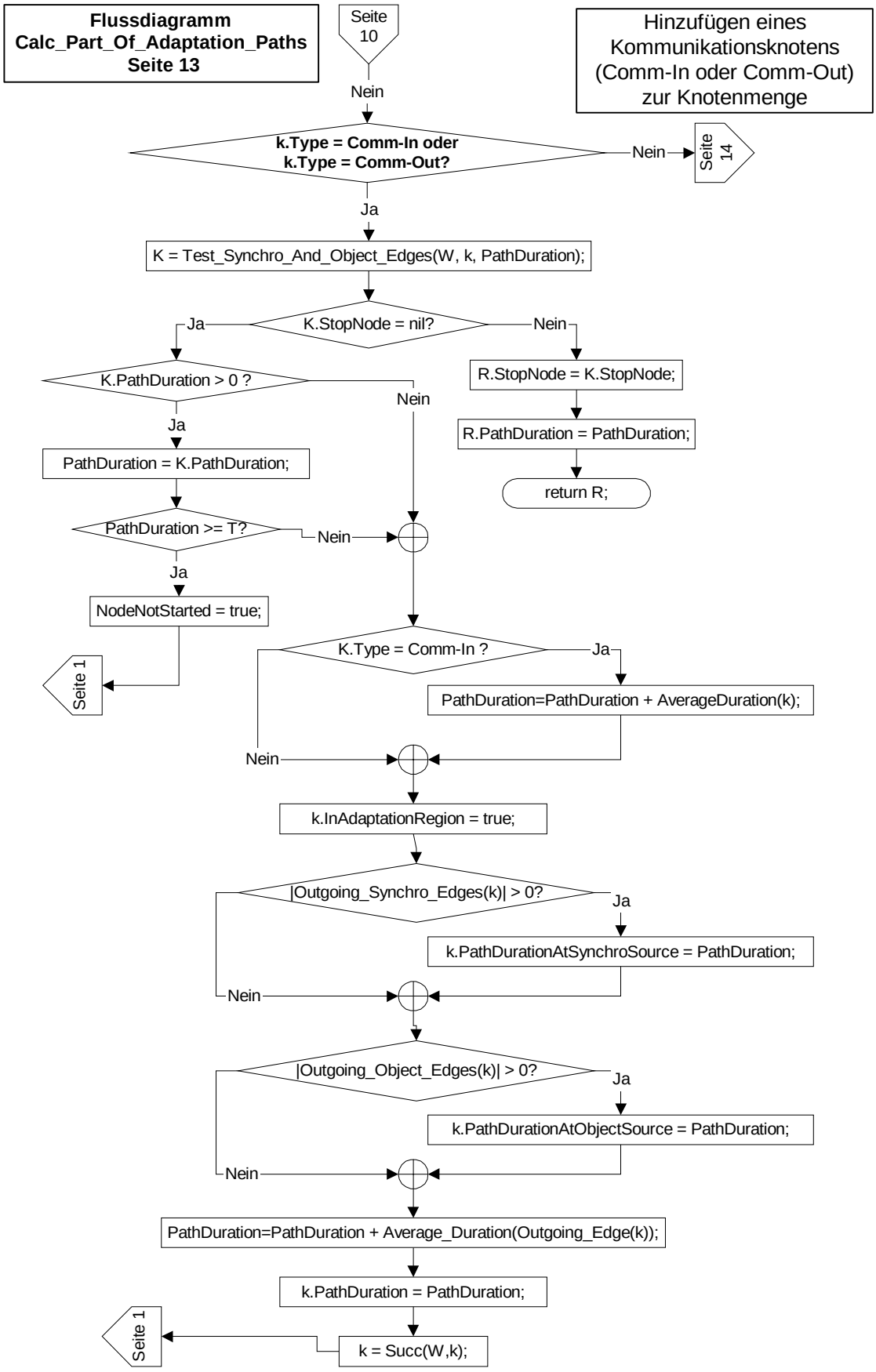

Flussdiagramm
Calc_Part_Of_Adaptation_Paths
Seite 8
Seite 6
Hinzufügen einer Split/ Join-Region zur Knotenmenge bei Unterbrechung vor Split-Knoten
DurationAtJoin > 0?
Nein
k = CorrespondingJoin;
PathDuration = T;
Seite 1
Ja
Type(CorrespondingJoin) = One-Join?
Nein
JoinNotReached = 0?
Nein
AllJoinNotReached = true;
k = CorrespondingJoin;
PathDuration = T;
Seite 1
Ja
Ja
PathDuration = DurationAtJoin;
K = Test_Synchro_And_Object_Edges(W, CorrespondingJoin, PathDuration);
K.StopNode = nil?
Ja
Seite 9
Nein
R.StopNode = K.StopNode;
R.PathDuration = PathDuration;
return R;

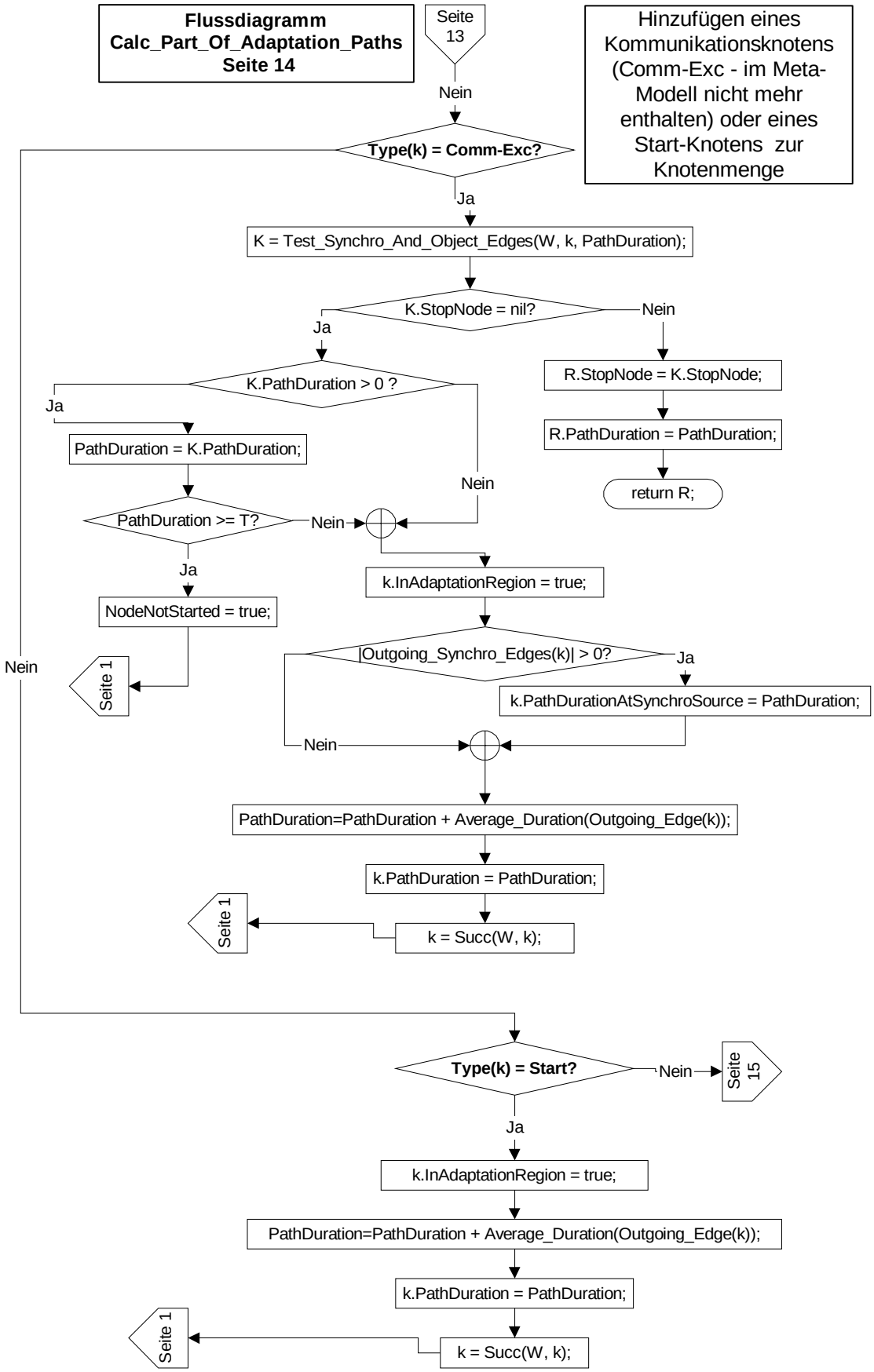

Flussdiagramm
Calc_Part_Of_Adaptation_Paths
Seite 9
Seite 8
Hinzufügen einer Split/ Join-Region zur Knotenmenge bei Unterbrechung vor Split-Knoten
Ja
K.PathDuration > 0 ?
Ja
PathDuration = K.PathDuration;
Nein
PathDuration >= T?
Nein
Ja
k = CorrespondingJoin;
NodeNotStarted = true;
Es gibt keine eingehende Synchronisationskante.
Seite 1
CorrespondingJoin.InAdaptationRegion = true;
|Outgoing_Synchro_Edges(CorrespondingJoin)| > 0?
Ja
CorrespondingJoin.PathDurationAtSynchroSource = PathDuration;
Nein
PathDuration=PathDuration + Average_Duration(Outgoing_Edge(CorrespondingJoin));
CorrespondingJoin.PathDuration = PathDuration;
Seite 1
k = Succ(W, CorrespondingJoin);

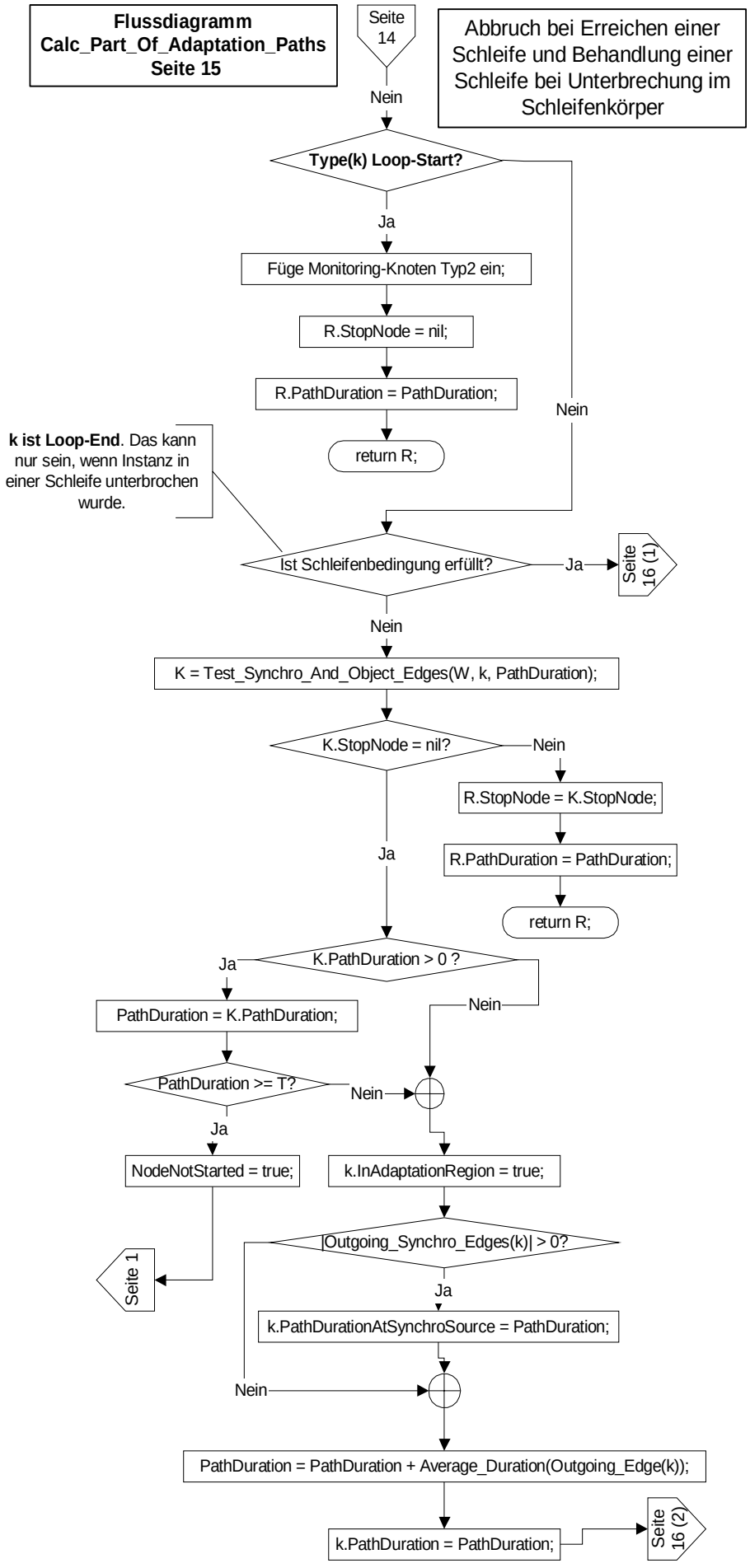

Flussdiagramm
Calc_Part_Of_Adaptation_Paths
Seite 10
Seite 5
Hinzufügen einer Split/Join-Region zur Knotenmenge bei Unterbrechung in der Verzweigung
Nein
Type(k) = Join?
Nein
Seite 13
Ja
k.RegionPartiallyCalculated = false?
Ja
Nein
DurationAtJoin = PathDuration;
DurationAtJoin = max(Pred(k).PathDuration) falls Type(k) = ALL-JOIN;
DurationAtJoin = min(Pred(k).PathDuration) falls Type(k) = ONE-JOIN;
Bed.: Pred(k).PathDuration > 0
SplitPaths = {node| node liegt auf Splitpfad und node ist in U (Unterbrechungsmenge der Instanz) oder bei node wurde schon einmal wegen einem fehlendem Vergleichswert für eine Synchrokante abgebrochen};
Seite 11
SplitPaths = {}?
Nein
Seite 11
Ja
StopNodesLocal = {}?
Ja
Seite 12
Nein
Für jeden Knoten n aus StopNodesLocal:
Sources[] = Sources_of_SynchroEdges(n);
i.PathDurationAtSynchroSource > 0 für alle Knoten i aus Sources?
Ja
SplitPaths = Union(SplitPaths, n);
Delete_From_List(n, StopNodesLocal);
Nein
Sources[i] liegen alle auf noch nicht abgeschlossenen Pfaden?
Ja
Nein
Füge Monitoring-Knoten Typ1 vor n ein;
Delete_From_List(n, StopNodesLocal);
Ende

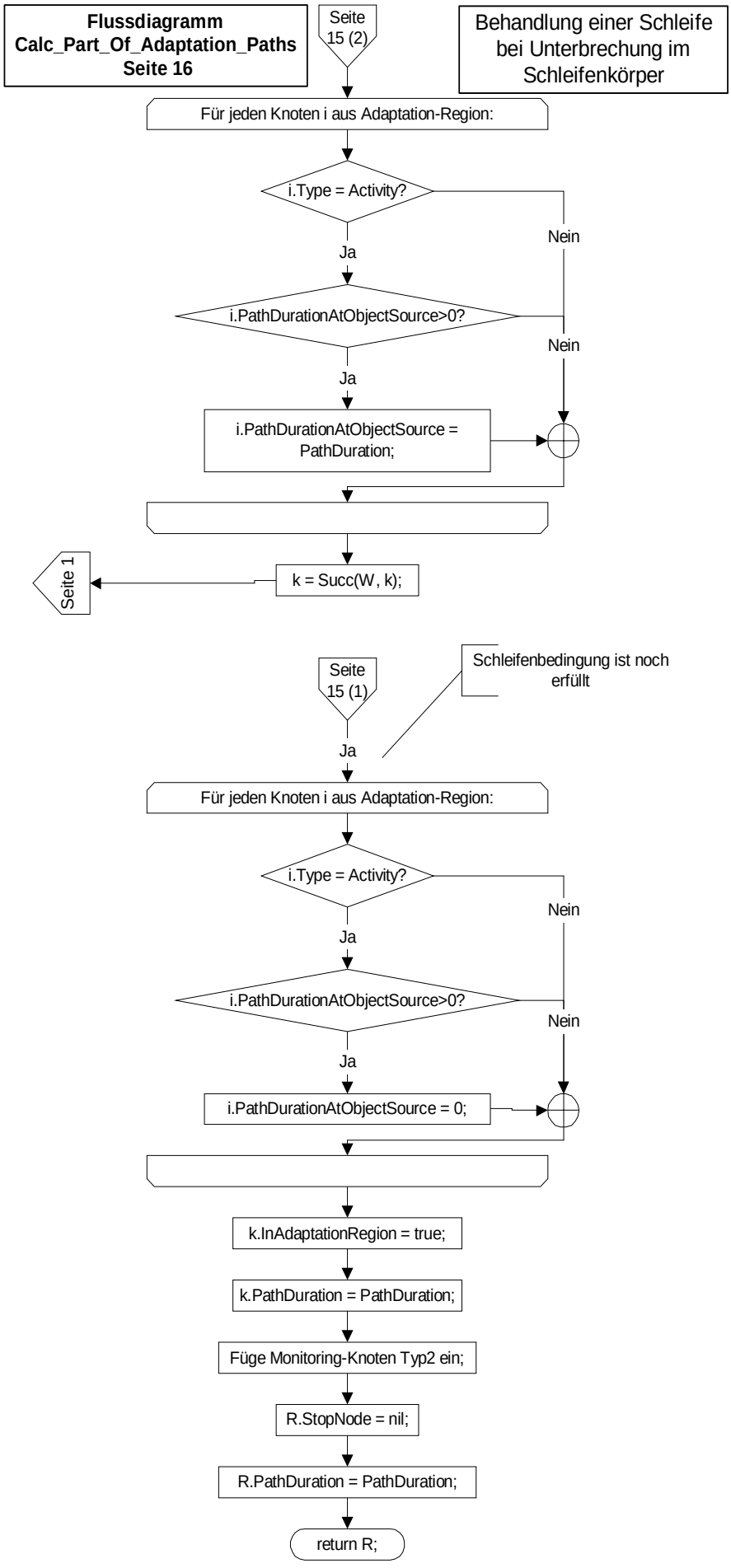

Flussdiagramm
Calc_Part_Of_Adaptation_Paths
Seite 11
Seite 10
Hinzufügen einer Split/Join-Region zur Knotenmenge bei Unterbrechung in der Verzweigung
SplitPathStartNode = First_Element(SplitPaths);
Delete_From_List(PathStartNode, SplitPaths);
S=Calc_Part_Of_Adaptation_Paths(W, SplitPathStartNode, T - SplitPathStartNode.PathDuration, Type(k));
S.SphereEnd = nil?
Ja
Nein
JoinNotReached++;
Type(S.SphereEnd) = Type(k)?
Nein
Source_Of_Synchro_Edge (S.SphereEnd) im gleichen Split?
Nein
k.RegionPartiallyCalculated = true;
R.StopNode = S.SphereEnd;
R.PathDuration = S.PathDuration;
return R;
Ja
Ja
StopNodesLocal= Union(StopNodesLocal, S.SphereEnd);
DurationAtJoin = max(DurationAtJoin, SplitPathStartNode.PathDuration + S.PathDuration) falls Type(k) = ALL-JOIN;
DurationAtJoin = min(DurationAtJoin, SplitPathStartNode.PathDuration + S.PathDuration) falls DurationAtJoin > 0
sonst
DurationAtJoin = SplitPathStartNode.PathDuration + S.PathDuration;
falls Type(k) = ONE-JOIN;
Seite 10


Flussdiagramm
Calc_Part_Of_Adaptation_Paths
Seite 12
Seite 10
Hinzufügen einer Split/Join-Region zur Knotenmenge bei Unterbrechung in der Verzweigung
Ja
Type(k) = One-Join?
Nein
Nein
JoinNotReached = 0?
Ja
Ja
AllJoinNotReached = true;
PathDuration = T;
Seite 1
PathDuration = DurationAtJoin;
K = Test_Synchro_And_Object_Edges(W, k, PathDuration);
K.StopNode = nil?
Nein
R.StopNode = K.StopNode;
R.PathDuration = PathDuration;
return R;
Ja
Ja
K.PathDuration > 0 ?
PathDuration = K.PathDuration;
Nein
PathDuration >= T?
Nein
Ja
NodeNotStarted = true;
Seite 1
k.InAdaptationRegion = true;
|Outgoing_Synchro_Edges(k)| > 0?
Ja
k.PathDurationAtSynchroSource = PathDuration;
Nein
PathDuration=PathDuration + Average_Duration(Outgoing_Edge(k));
k.PathDuration = PathDuration;
Seite 1
k = Succ(W, k);

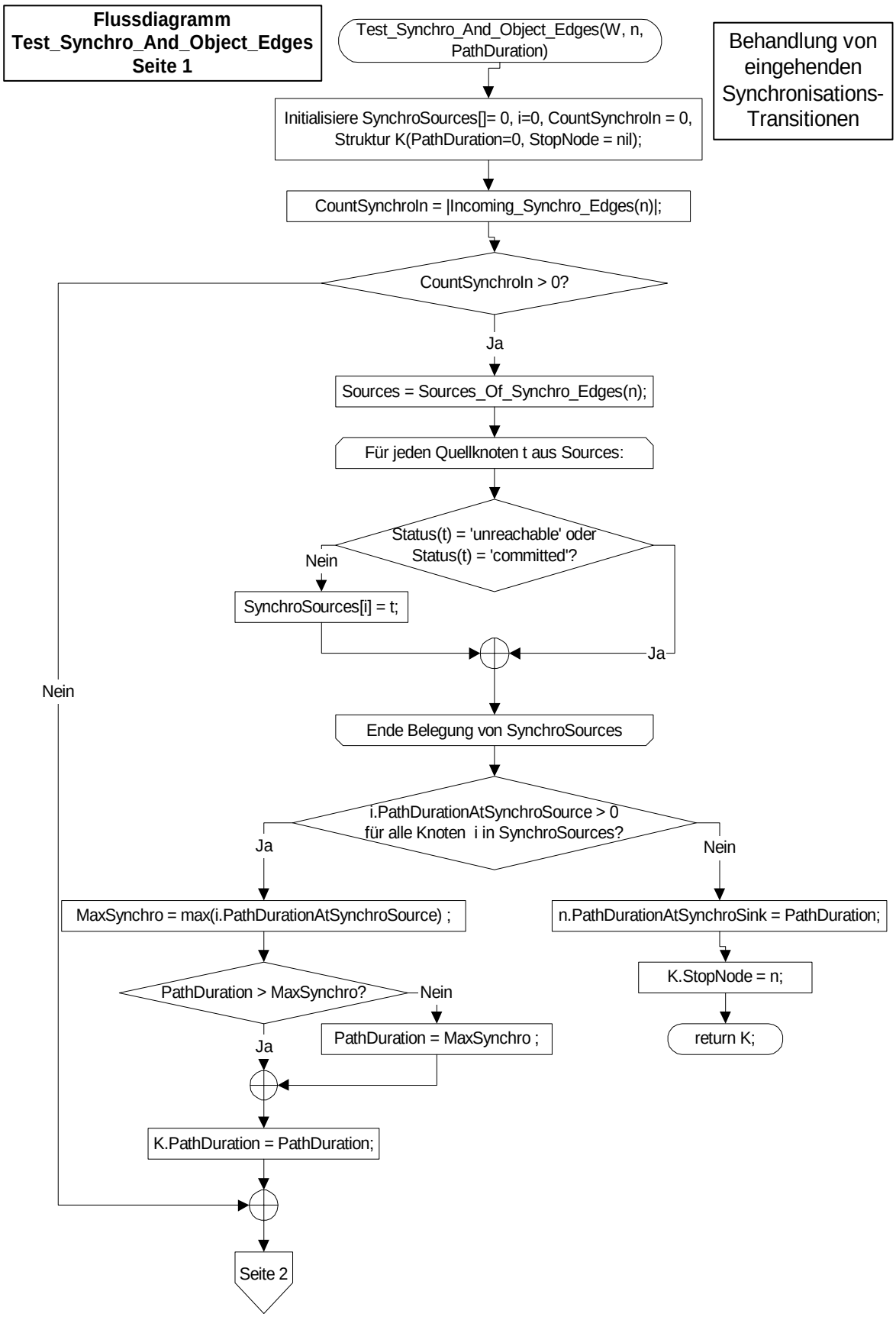

Flussdiagramm
Calc_Part_Of_Adaptation_Paths
Seite 13
Seite 10
Hinzufügen eines Kommunikationsknotens (Comm-In oder Comm-Out) zur Knotenmenge
Nein
k.Type = Comm-In oder k.Type = Comm-Out?
Nein
Seite 14
Ja
K = Test_Synchro_And_Object_Edges(W, k, PathDuration);
Ja
K.StopNode = nil?
Nein
K.PathDuration > 0 ?
Nein
R.StopNode = K.StopNode;
R.PathDuration = PathDuration;
return R;
Ja
PathDuration = K.PathDuration;
PathDuration >= T?
Nein
Ja
NodeNotStarted = true;
Seite 1
K.Type = Comm-In ?
Ja
PathDuration=PathDuration + AverageDuration(k);
Nein
k.InAdaptationRegion = true;
|Outgoing_Synchro_Edges(k)| > 0?
Ja
k.PathDurationAtSynchroSource = PathDuration;
Nein
|Outgoing_Object_Edges(k)| > 0?
Ja
k.PathDurationAtObjectSource = PathDuration;
Nein
PathDuration=PathDuration + Average_Duration(Outgoing_Edge(k));
k.PathDuration = PathDuration;
Seite 1
k = Succ(W,k);

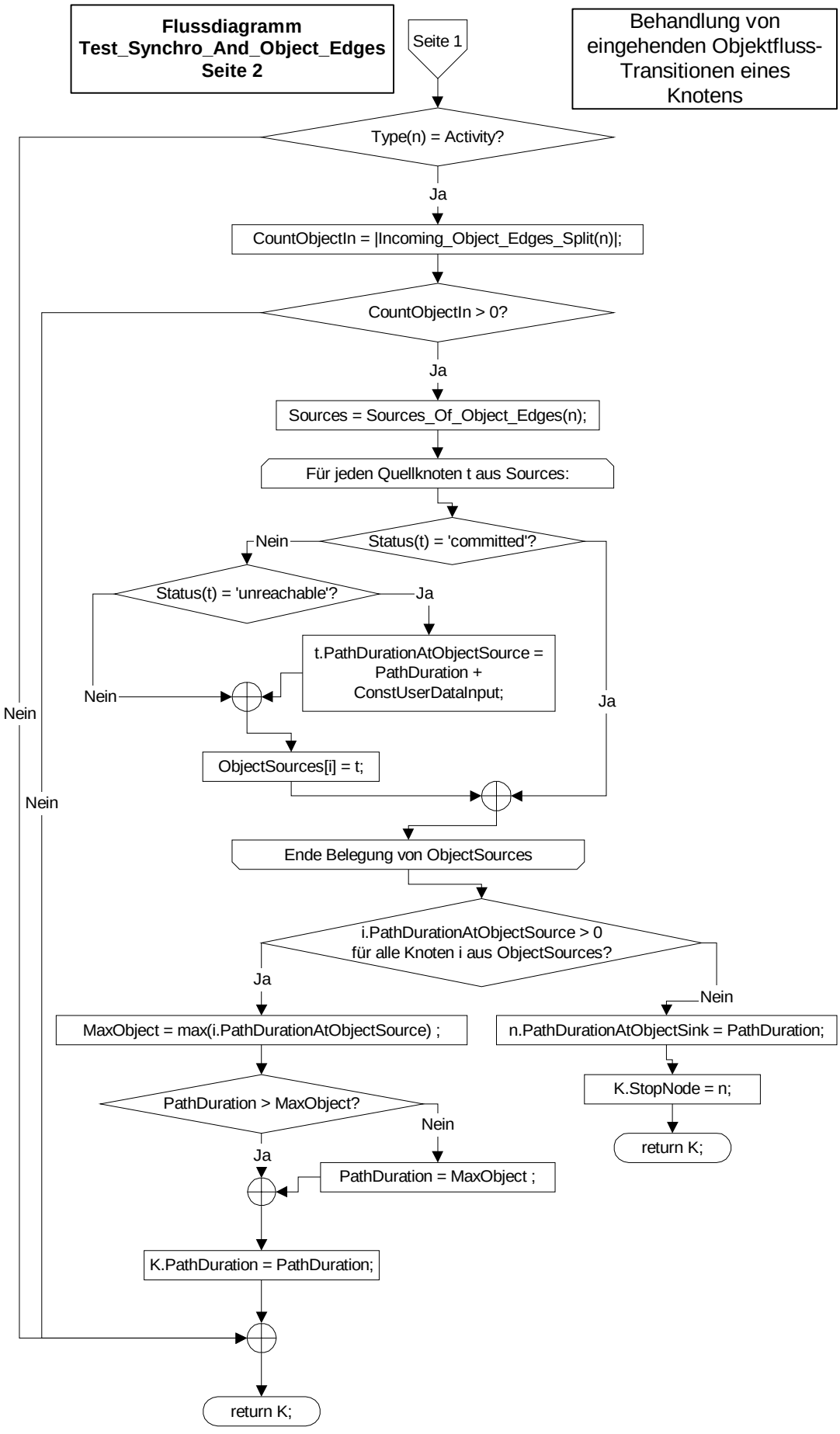

Flussdiagramm
Calc_Part_Of_Adaptation_Paths
Seite 14
Seite 13
Nein
Type(k) = Comm-Exc?
Hinzufügen eines Kommunikationsknotens (Comm-Exc - im Meta-Modell nicht mehr enthalten) oder eines Start-Knotens zur Knotenmenge
Ja
K = Test_Synchro_And_Object_Edges(W, k, PathDuration);
K.StopNode = nil?
Nein
Ja
R.StopNode = K.StopNode;
R.PathDuration = PathDuration;
return R;
K.PathDuration > 0 ?
Ja
Nein
PathDuration = K.PathDuration;
PathDuration >= T?
Nein
Ja
NodeNotStarted = true;
Seite 1
k.InAdaptationRegion = true;
|Outgoing_Synchro_Edges(k)| > 0?
Ja
k.PathDurationAtSynchroSource = PathDuration;
Nein
PathDuration=PathDuration + Average_Duration(Outgoing_Edge(k));
k.PathDuration = PathDuration;
Seite 1
k = Succ(W, k);
Nein
Type(k) = Start?
Nein
Seite 15
Ja
k.InAdaptationRegion = true;
PathDuration=PathDuration + Average_Duration(Outgoing_Edge(k));
k.PathDuration = PathDuration;
Seite 1
k = Succ(W, k);


Flussdiagramm
Calc_Part_Of_Adaptation_Paths
Seite 15
Seite 14
Abbruch bei Erreichen einer Schleife und Behandlung einer Schleife bei Unterbrechung im Schleifenkörper
Nein
Type(k) Loop-Start?
Ja
Füge Monitoring-Knoten Typ2 ein;
R.StopNode = nil;
R.PathDuration = PathDuration;
return R;
Nein
k ist Loop-End. Das kann nur sein, wenn Instanz in einer Schleife unterbrochen wurde.
Ist Schleifenbedingung erfüllt?
Ja
Seite 16 (1)
Nein
K = Test_Synchro_And_Object_Edges(W, k, PathDuration);
K.StopNode = nil?
Nein
R.StopNode = K.StopNode;
R.PathDuration = PathDuration;
return R;
Ja
K.PathDuration > 0 ?
Ja
Nein
PathDuration = K.PathDuration;
PathDuration >= T?
Nein
Ja
NodeNotStarted = true;
k.InAdaptationRegion = true;
Seite 1
|Outgoing_Synchro_Edges(k)| > 0?
Ja
k.PathDurationAtSynchroSource = PathDuration;
Nein
PathDuration = PathDuration + Average_Duration(Outgoing_Edge(k));
k.PathDuration = PathDuration;
Seite 16 (2)

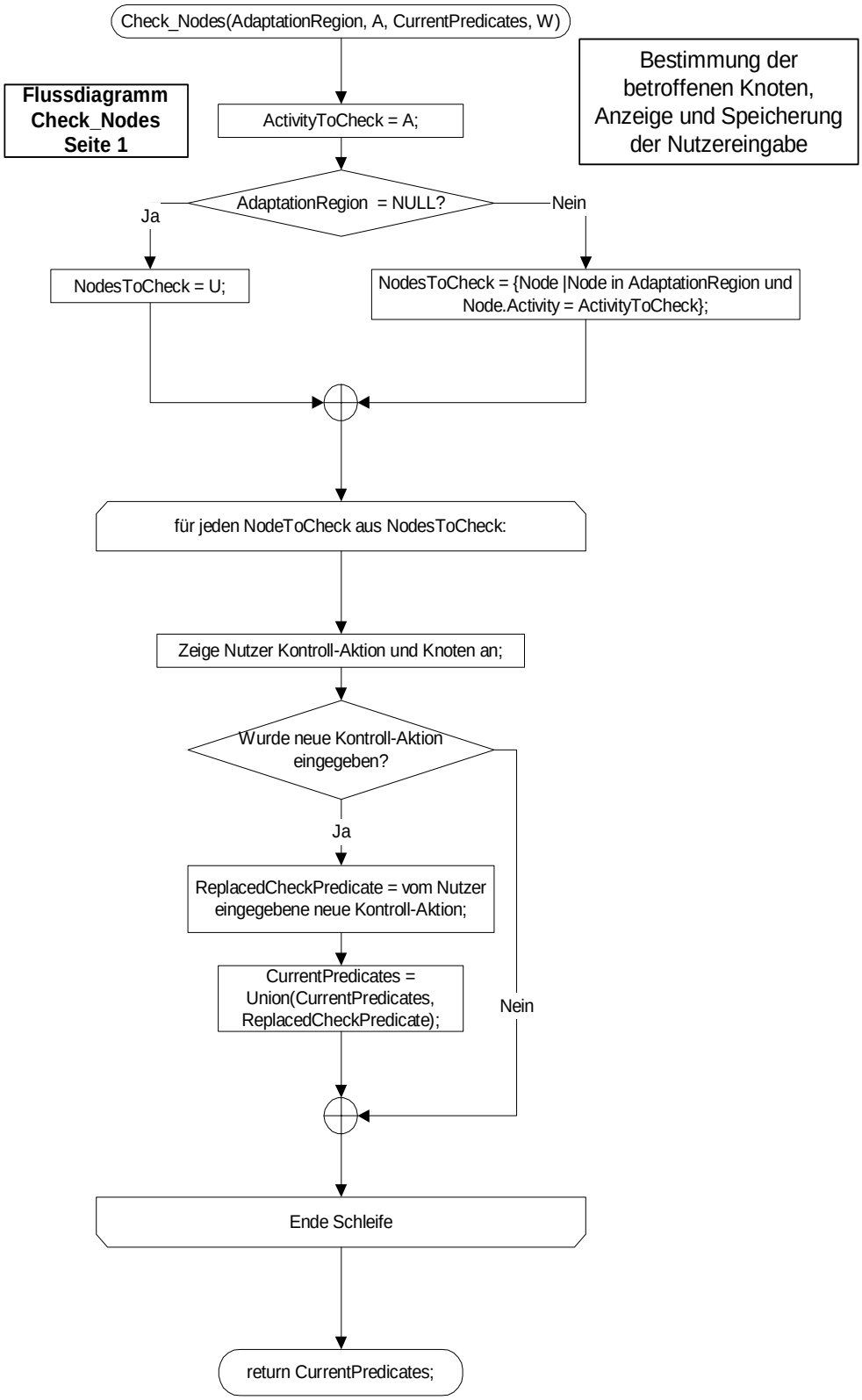

Flussdiagramm
Calc_Part_Of_Adaptation_Paths
Seite 16
Seite
15 (2)
Behandlung einer Schleife
bei Unterbrechung im
Schleifenkörper
Für jeden Knoten i aus Adaptation-Region:
i.Type = Activity?
Nein
Ja
i.PathDurationAtObjectSource>0?
Nein
Ja
i.PathDurationAtObjectSource =
PathDuration;
Seite 1
k = Succ(W, k);
Seite
15 (1)
Schleifenbedingung ist noch
erfüllt
Ja
Für jeden Knoten i aus Adaptation-Region:
i.Type = Activity?
Nein
Ja
i.PathDurationAtObjectSource>0?
Nein
Ja
i.PathDurationAtObjectSource = 0;
k.InAdaptationRegion = true;
k.PathDuration = PathDuration;
Füge Monitoring-Knoten Typ2 ein;
R.StopNode = nil;
R.PathDuration = PathDuration;
return R;

B5. Berücksichtigung von Synchronisations- und Objektfluss-Transitionen

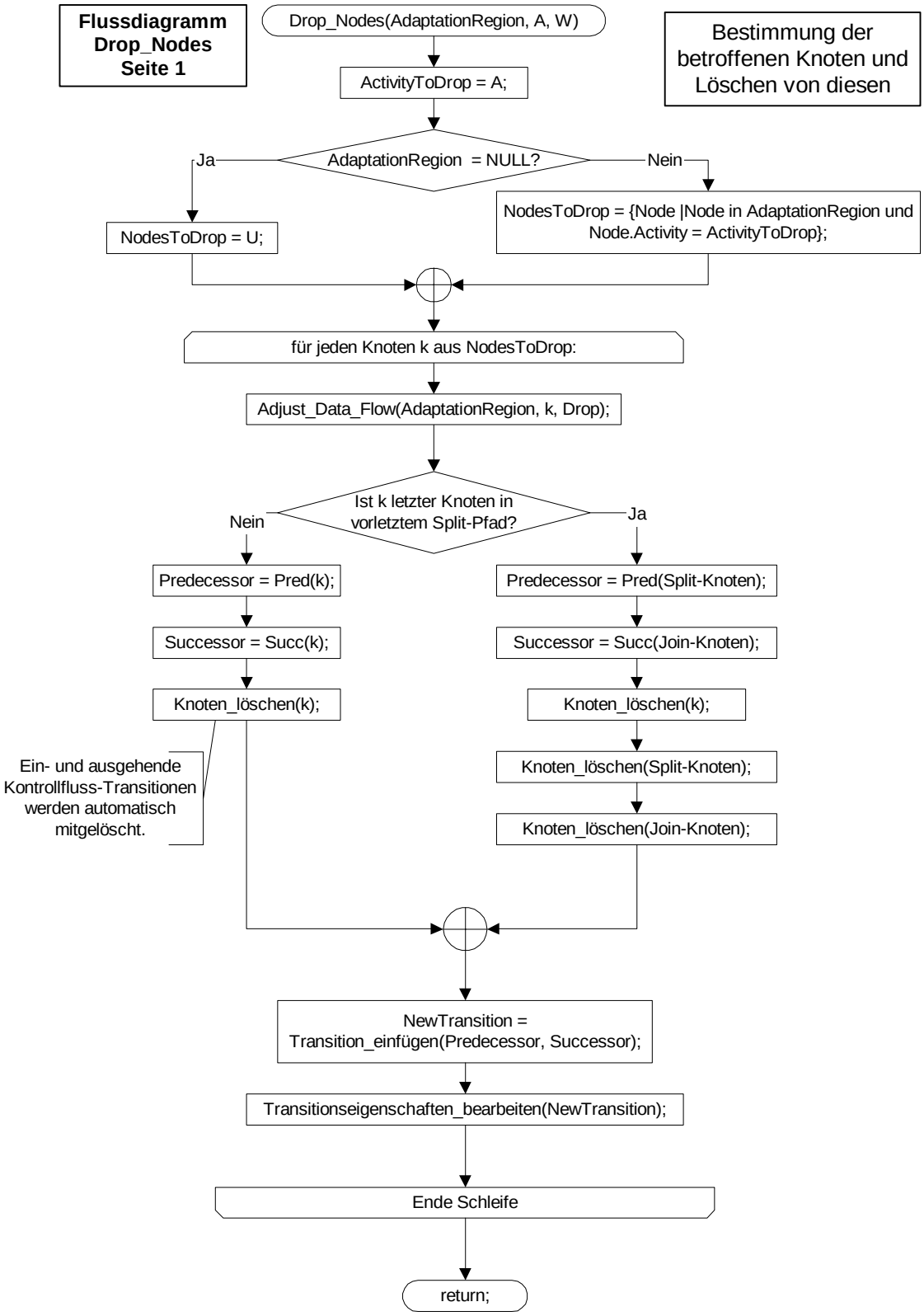



Flussdiagramm
Test_Synchro_And_Object_Edges
Seite 2
Seite 1
Behandlung von eingehenden Objektfluss-Transitionen eines Knotens
Type(n) = Activity?
Ja
CountObjectIn = |Incoming_Object_Edges_Split(n)|;
CountObjectIn > 0?
Ja
Sources = Sources_Of_Object_Edges(n);
Für jeden Quellknoten t aus Sources:
Nein
Status(t) = 'committed'?
Status(t) = 'unreachable'?
Ja
t.PathDurationAtObjectSource = PathDuration + ConstUserDataInput;
Nein
Nein
Ja
ObjectSources[i] = t;
Nein
Ende Belegung von ObjectSources
i.PathDurationAtObjectSource > 0 für alle Knoten i aus ObjectSources?
Ja
Nein
MaxObject = max(i.PathDurationAtObjectSource) ;
n.PathDurationAtObjectSink = PathDuration;
PathDuration > MaxObject?
K.StopNode = n;
Nein
return K;
Ja
PathDuration = MaxObject ;
K.PathDuration = PathDuration;
return K;

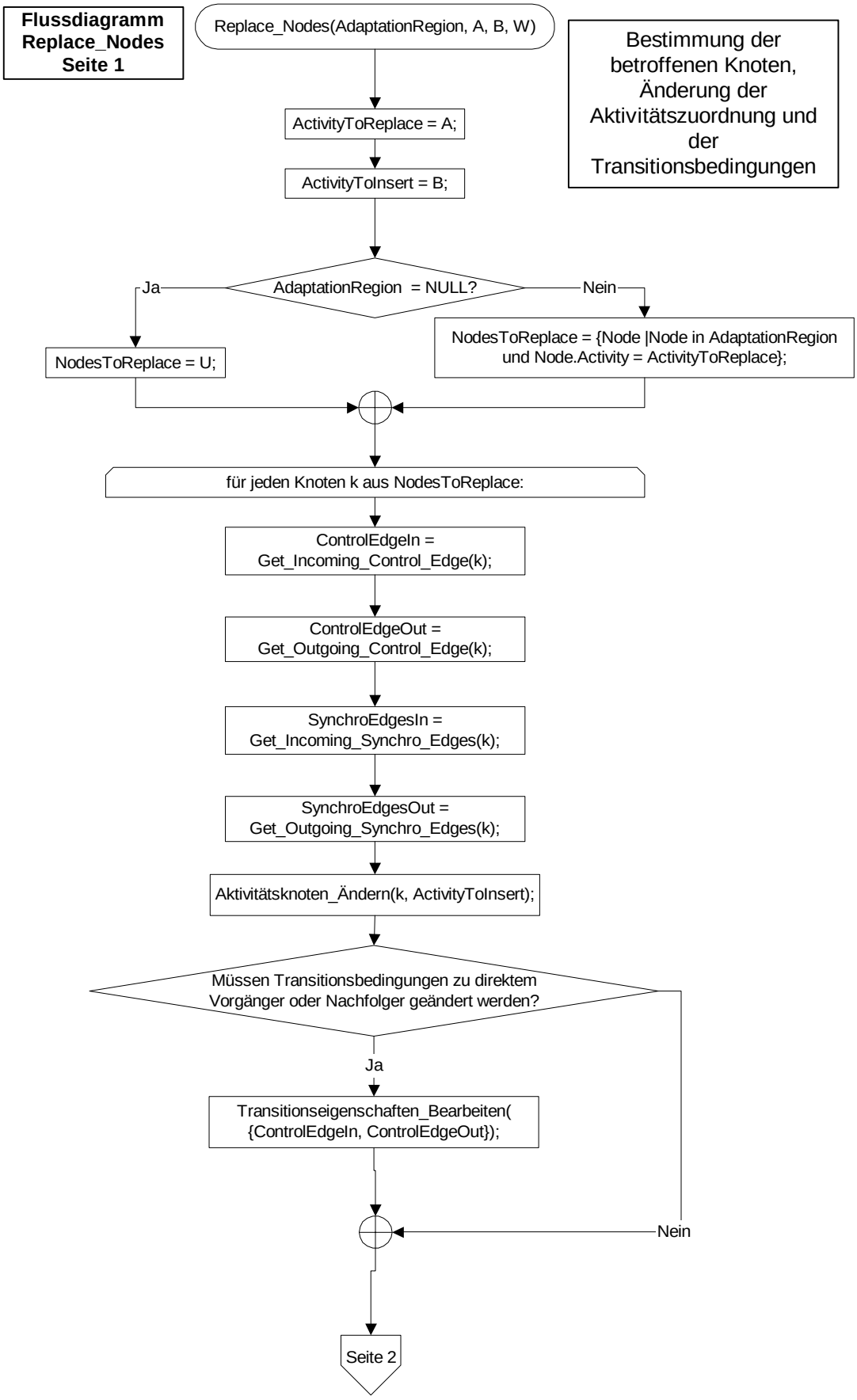
B.6 Kontroll-Aktion *check*(A)

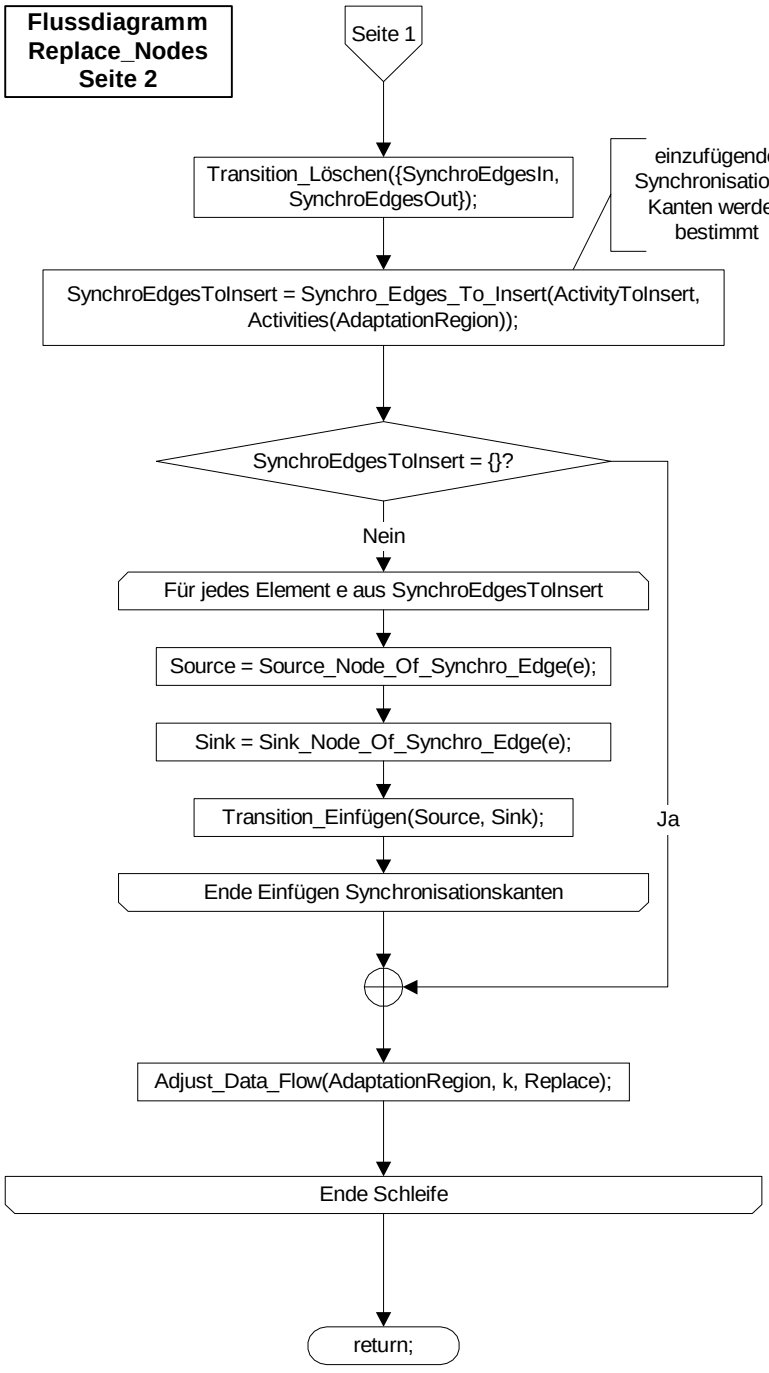


B.7 Kontroll-Aktion *drop*(A)

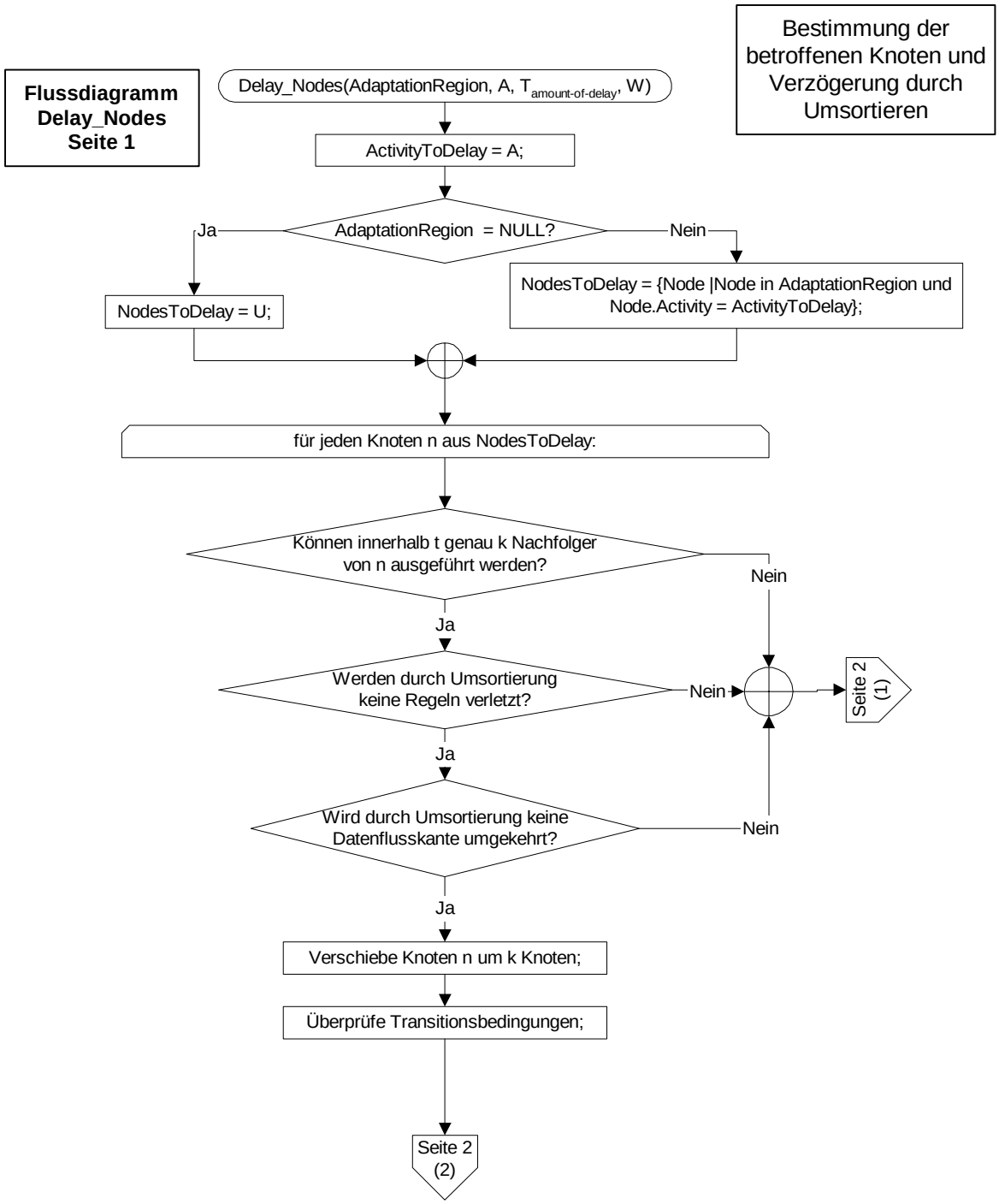


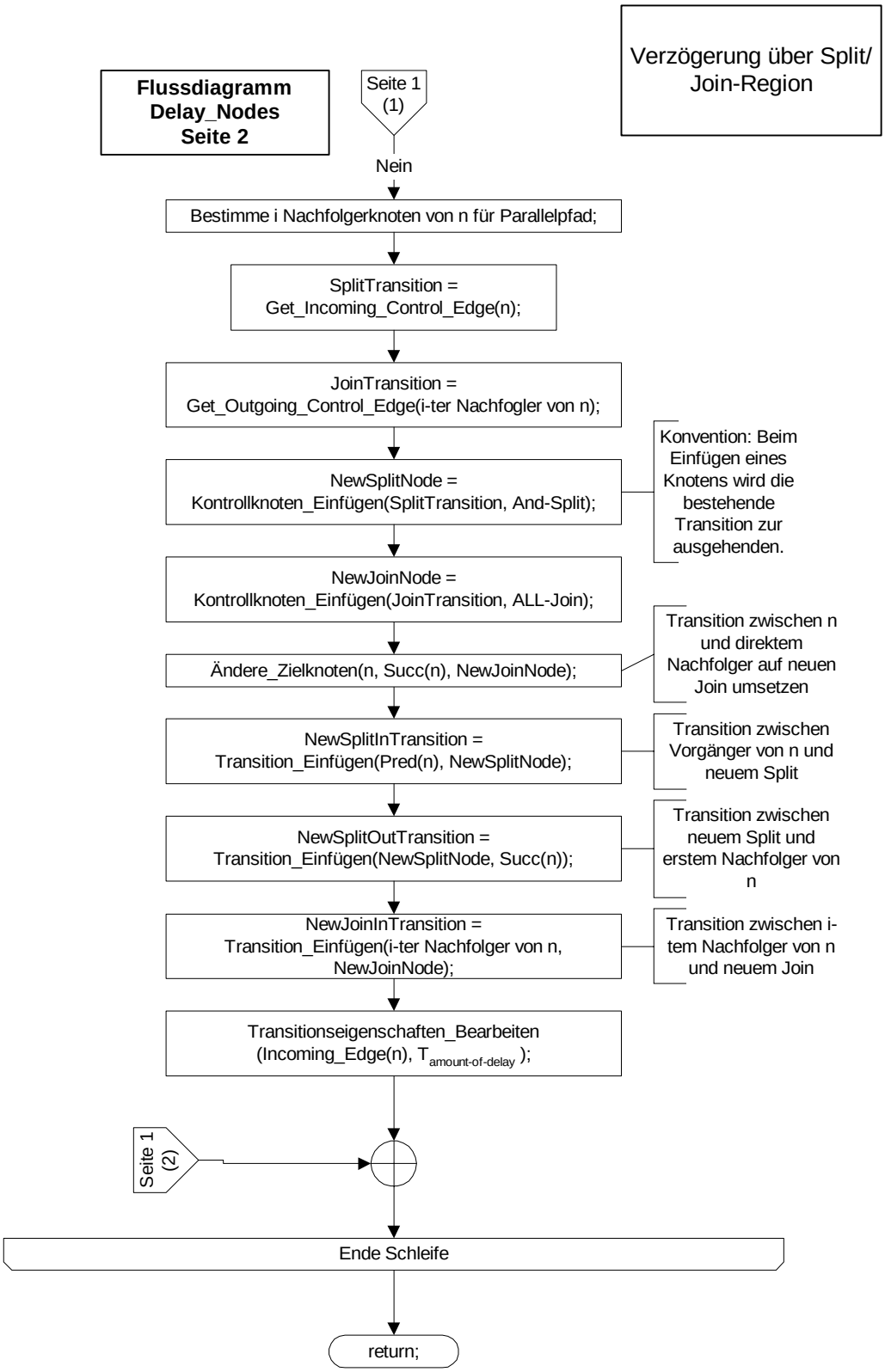
B.8 Kontroll-Aktion *replace*(A, B)



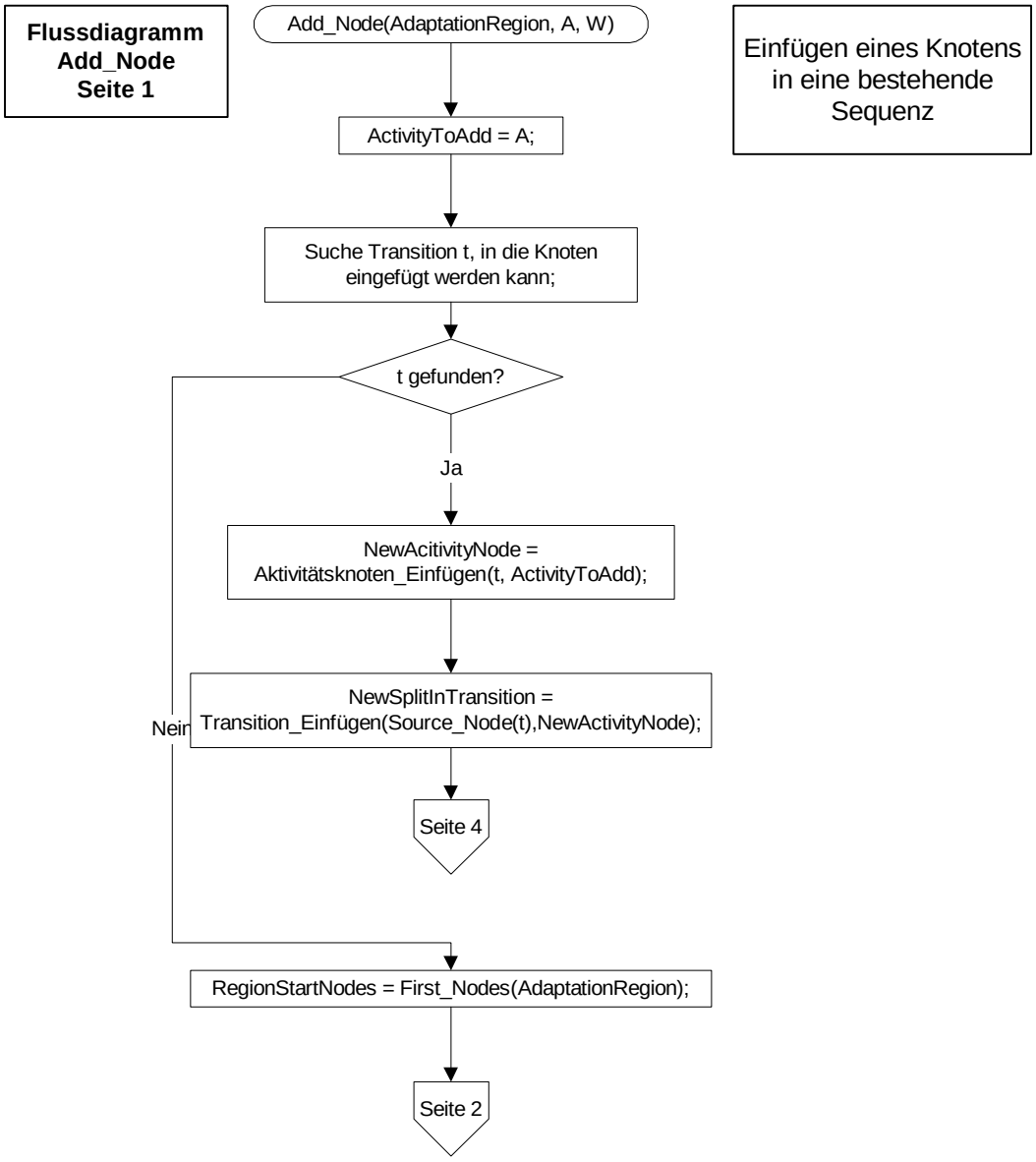

Flussdiagramm
Replace_Nodes
Seite 2
Seite 1
Transition_Löschen({SynchroEdgesIn, SynchroEdgesOut});
einzufügende Synchronisations-Kanten werden bestimmt
SynchroEdgesToInsert = Synchro_Edges_To_Insert(ActivityToInsert, Activities(AdaptationRegion));
SynchroEdgesToInsert = {}?
Nein
Ja
Für jedes Element e aus SynchroEdgesToInsert
Source = Source_Node_Of_Synchro_Edge(e);
Sink = Sink_Node_Of_Synchro_Edge(e);
Transition_Einfügen(Source, Sink);
Ende Einfügen Synchronisationskanten
Adjust_Data_Flow(AdaptationRegion, k, Replace);
Ende Schleife
return;

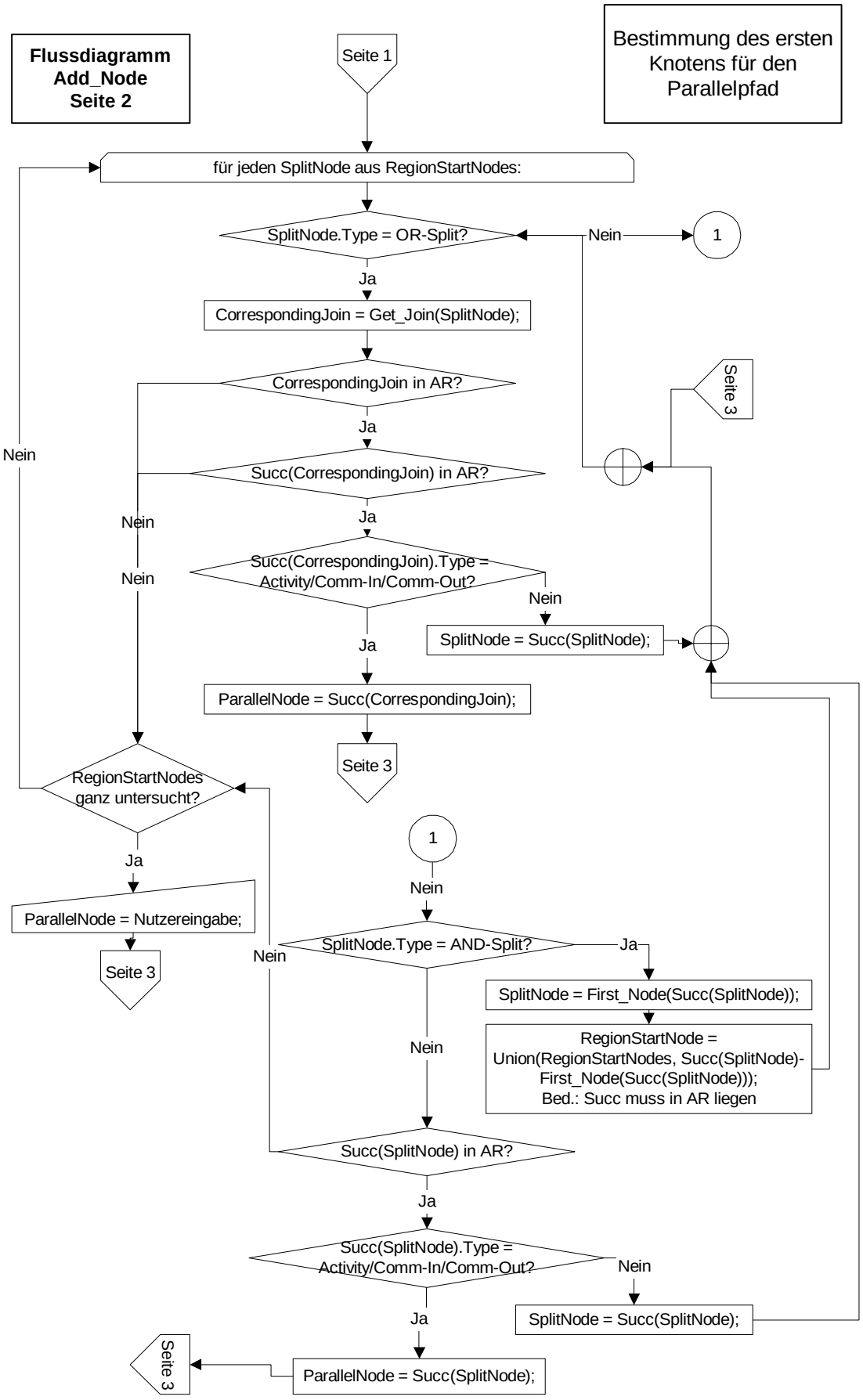
B.9 Kontroll-Aktion *delay*(A, t)

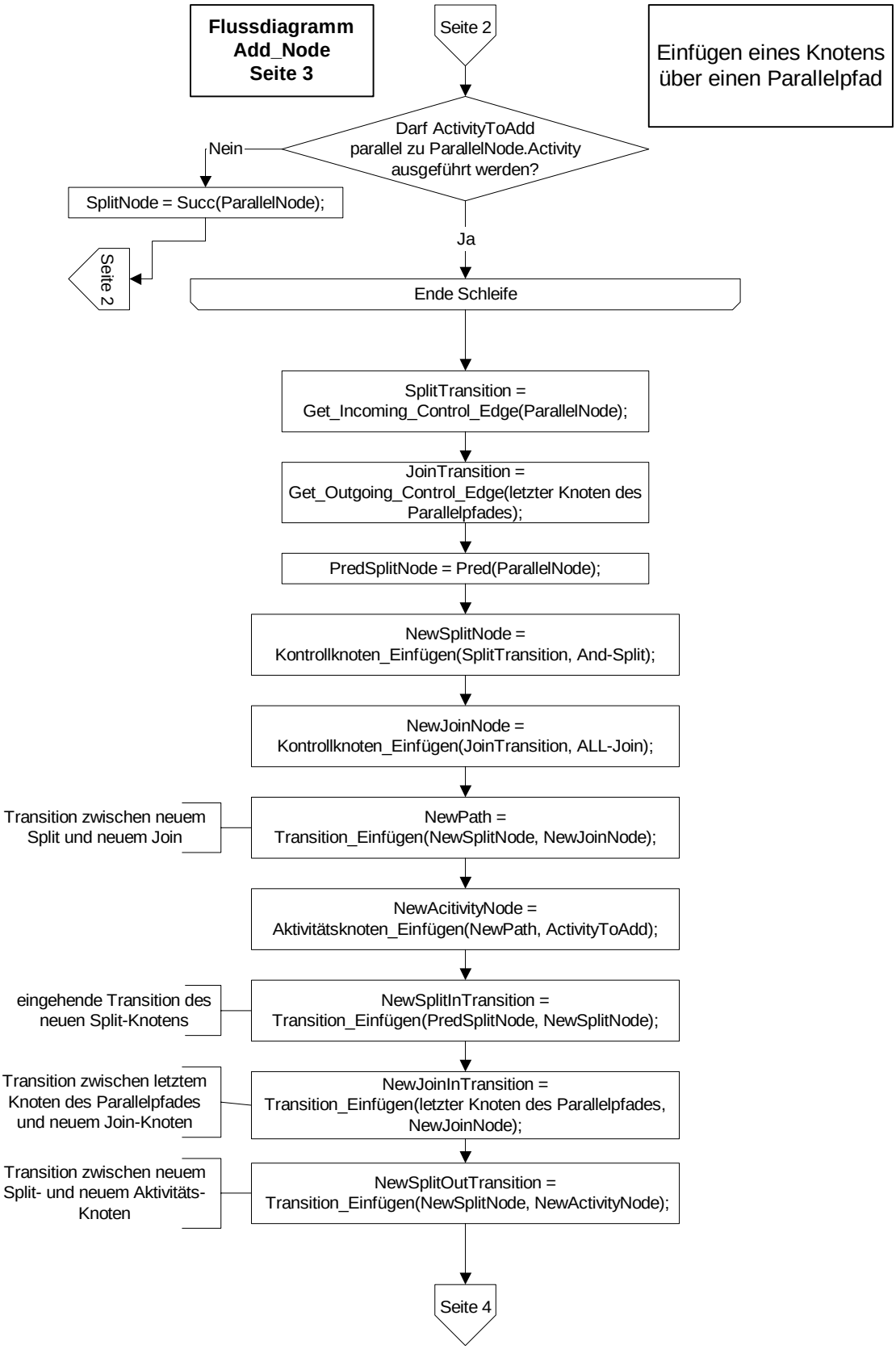


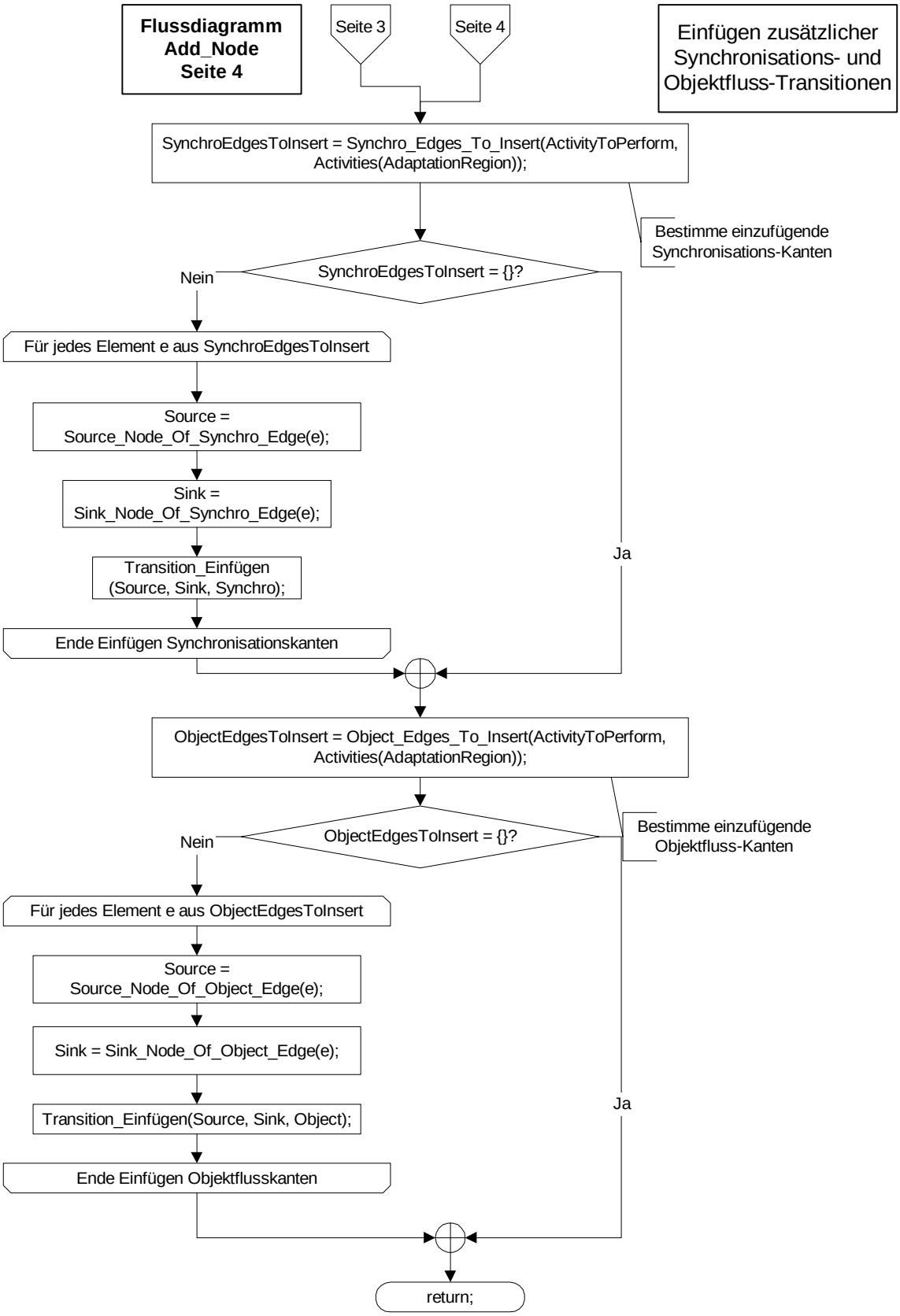

Verzögerung über Split/ Join-Region
Flussdiagramm Delay_Nodes Seite 2
Seite 1 (1)
Nein
Bestimme i Nachfolgerknoten von n für Parallelpfad;
SplitTransition = Get_Incoming_Control_Edge(n);
JoinTransition = Get_Outgoing_Control_Edge(i-ter Nachfogler von n);
NewSplitNode = Kontrollknoten_Einfügen(SplitTransition, And-Split);
Konvention: Beim Einfügen eines Knotens wird die bestehende Transition zur ausgehenden.
NewJoinNode = Kontrollknoten_Einfügen(JoinTransition, ALL-Join);
Ändere_Zielknoten(n, Succ(n), NewJoinNode);
Transition zwischen n und direktem Nachfolger auf neuen Join umsetzen
NewSplitInTransition = Transition_Einfügen(Pred(n), NewSplitNode);
Transition zwischen Vorgänger von n und neuem Split
NewSplitOutTransition = Transition_Einfügen(NewSplitNode, Succ(n));
Transition zwischen neuem Split und erstem Nachfolger von n
NewJoinInTransition = Transition_Einfügen(i-ter Nachfolger von n, NewJoinNode);
Transition zwischen i-tem Nachfolger von n und neuem Join
Transitionseigenschaften_Bearbeiten (Incoming_Edge(n), T amount-of-delay);
Seite 1 (2)
Ende Schleife
return;

B.10 Kontroll-Aktion *add*(A)




Flussdiagramm
Add_Node
Seite 2
Bestimmung des ersten Knotens für den Parallelpfad
Seite 1
für jeden SplitNode aus RegionStartNodes:
SplitNode.Type = OR-Split?
Nein
1
Ja
CorrespondingJoin = Get_Join(SplitNode);
CorrespondingJoin in AR?
Seite 3
Ja
Nein
Succ(CorrespondingJoin) in AR?
Nein
Ja
Nein
Succ(CorrespondingJoin).Type = Activity/Comm-In/Comm-Out?
Nein
SplitNode = Succ(SplitNode);
Ja
ParallelNode = Succ(CorrespondingJoin);
Seite 3
RegionStartNodes ganz untersucht?
1
Ja
Nein
ParallelNode = Nutzereingabe;
Seite 3
SplitNode.Type = AND-Split?
Ja
Nein
SplitNode = First_Node(Succ(SplitNode));
RegionStartNode = Union(RegionStartNodes, Succ(SplitNode)-First_Node(Succ(SplitNode)));
Bed.: Succ muss in AR liegen
Nein
Succ(SplitNode) in AR?
Ja
Succ(SplitNode).Type = Activity/Comm-In/Comm-Out?
Nein
SplitNode = Succ(SplitNode);
Ja
Seite 3
ParallelNode = Succ(SplitNode);


Flussdiagramm
Add_Node
Seite 3
Seite 2
Einfügen eines Knotens
über einen Parallelpfad
Darf ActivityToAdd
parallel zu ParallelNode.Activity
ausgeführt werden?
Nein
Ja
SplitNode = Succ(ParallelNode);
Seite 2
Ende Schleife
SplitTransition =
Get_Incoming_Control_Edge(ParallelNode);
JoinTransition =
Get_Outgoing_Control_Edge(letzter Knoten des
Parallelpfades);
PredSplitNode = Pred(ParallelNode);
NewSplitNode =
Kontrollknoten_Einfügen(SplitTransition, And-Split);
NewJoinNode =
Kontrollknoten_Einfügen(JoinTransition, ALL-Join);
Transition zwischen neuem
Split und neuem Join
NewPath =
Transition_Einfügen(NewSplitNode, NewJoinNode);
NewAcitivityNode =
Aktivitätsknoten_Einfügen(NewPath, ActivityToAdd);
eingehende Transition des
neuen Split-Knotens
NewSplitInTransition =
Transition_Einfügen(PredSplitNode, NewSplitNode);
Transition zwischen letztem
Knoten des Parallelpfades
und neuem Join-Knoten
NewJoinInTransition =
Transition_Einfügen(letzter Knoten des Parallelpfades,
NewJoinNode);
Transition zwischen neuem
Split- und neuem Aktivitäts-
Knoten
NewSplitOutTransition =
Transition_Einfügen(NewSplitNode, NewActivityNode);
Seite 4


Flussdiagramm
Add_Node
Seite 4
Seite 3
Seite 4
Einfügen zusätzlicher Synchronisations- und Objektfluss-Transitionen
SynchroEdgesToInsert = Synchro_Edges_To_Insert(ActivityToPerform, Activities(AdaptationRegion));
Bestimme einzufügende Synchronisations-Kanten
SynchroEdgesToInsert = {}?
Nein
Ja
Für jedes Element e aus SynchroEdgesToInsert
Source = Source_Node_Of_Synchro_Edge(e);
Sink = Sink_Node_Of_Synchro_Edge(e);
Transition_Einfügen (Source, Sink, Synchro);
Ende Einfügen Synchronisationskanten
ObjectEdgesToInsert = Object_Edges_To_Insert(ActivityToPerform, Activities(AdaptationRegion));
Bestimme einzufügende Objektfluss-Kanten
ObjectEdgesToInsert = {}?
Nein
Ja
Für jedes Element e aus ObjectEdgesToInsert
Source = Source_Node_Of_Object_Edge(e);
Sink = Sink_Node_Of_Object_Edge(e);
Transition_Einfügen(Source, Sink, Object);
Ende Einfügen Objektflusskanten
return;

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Leipzig, Juli 2000 _____

Unterschrift