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

Effiziente Realisierung unternehmensweiter Workflow-Management-Systeme

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

vorgelegt von

THOMAS BAUER
aus Laupheim

Oktober 2000

Amtierender Dekan: Prof. Dr. G. Palm

Gutachter: Prof. Dr. P. Dadam
Prof. Dr. M. Weber

Tag der Promotion: 8. Februar 2001

Vorwort

Die vorliegende Dissertation entstand während meiner Tätigkeit als Wissenschaftlicher Mitarbeiter in der Abteilung Datenbanken und Informationssysteme der Universität Ulm. Die Arbeit ist im ADEPT-Projekt angesiedelt, an dem in der Abteilung seit 1995 gearbeitet wird. Ein Ziel des Projektes ist es, ein Workflow-Management-System zu entwickeln, das die Basis für anspruchsvolle prozessorientierte Anwendungen bilden kann. Deshalb wurden im Laufe der Zeit zahlreiche Aspekte untersucht, die von heutigen Systemen noch nicht adäquat unterstützt werden.

Mein persönlicher Dank gilt meinem Betreuer Prof. Dr. Peter Dadam, der mich während meiner Promotionszeit tatkräftig unterstützt hat. In zahlreichen konstruktiven Diskussionen hat er zum inhaltlichen Gelingen der Arbeit beigetragen. Darüber hinaus habe ich von seiner problemorientierten Sichtweise stark profitiert. Sein stetes Hinterfragen des Nutzens, der praktischen Relevanz und der Darstellung jedes Aspekts hat entscheidend zum Gelingen meiner wissenschaftlichen Veröffentlichungen und nicht zuletzt auch der vorliegenden Arbeit beigetragen.

Ebenso möchte ich Prof. Dr. Michael Weber herzlich danken. Er hat sich zum einen als Gutachter zur Verfügung gestellt, zum anderen waren die fachlichen Gespräche mit ihm äußerst angenehm und motivierend. Insbesondere danke ich ihm für seine zahlreichen Hinweise auf verwandte Themen aus dem Bereich der verteilten Systeme.

Ein ganz besonderer Dank gilt meinem Kollegen Manfred Reichert. Die langen, äußerst konstruktiven Diskussionen mit ihm haben mich stets auf neue Problemstellungen aufmerksam gemacht und mir Ideen zur Entwicklung von neuen Lösungsansätzen geliefert. Ich danke ihm außerdem für die Unterstützung, die er mir bei der Erstellung meiner wissenschaftlichen Veröffentlichungen zukommen ließ. Den weiteren Mitgliedern des ADEPT-Teams, Clemens Hensinger und Thomas Strzeletz, möchte ich herzlich für die vielen fachlichen Diskussionen danken, die sie mit mir geführt haben. Allen anderen Mitgliedern der Abteilung danke ich dafür, dass sie stets für ein angenehmes Arbeitsumfeld und gute Stimmung gesorgt haben.

Der größte Dank gebührt jedoch meiner Freundin Kristina. Sie hat durch ihre Liebe und Unterstützung das Entstehen dieser Arbeit erst möglich gemacht.

Ulm, im Oktober 2000

Kurzfassung

Workflow-Management-Systeme ermöglichen die computergestützte Abwicklung von Geschäftsprozessen. Allerdings sind die Einsatzmöglichkeiten heutiger Systeme wegen ihrer unzureichenden Skalierbarkeit bei hoher Last noch stark eingeschränkt. Dies ist insbesondere bei unternehmensweiten Anwendungssystemen problematisch, weil hier typischerweise sehr viele Benutzer mit dem System arbeiten und die verschiedenen Teilnetze evtl. nur durch langsame Kommunikationsverbindungen miteinander vernetzt sind. Dies kann zu einer Überlastung der Workflow-Server und des Kommunikationssystems führen. Das Ziel dieser Arbeit ist die Entwicklung eines effizienten Verfahrens für verteiltes Workflow-Management, das sich dadurch auszeichnet, dass weder ein Workflow-Server noch eine Komponente des Kommunikationsnetzwerks überlastet wird. Es muss außerdem im Zusammenspiel mit weiteren fortschrittlichen Workflow-Konzepten (z.B. Zurücksetzen oder dynamische Änderung von Workflows, komplexe Bearbeiterzuordnungen) funktionieren, da diese benötigt werden, um das System für ein breites Spektrum von Anwendungen einsetzen zu können.

In ADEPT$_{distribution}$ findet die Workflow-Steuerung verteilt statt. Workflow-Instanzen werden abschnittsweise durch verschiedene Workflow-Server kontrolliert. Dabei wird das Ziel verfolgt, den Workflow-Server einer Aktivität nahe bei ihren potentiellen Bearbeitern zu wählen, was die entstehende Kommunikationslast reduziert. Um eine solch verteilte WF-Ausführung zu ermöglichen, sind Migrationen notwendig. Bei diesen wird eine Workflow-Instanz von einem Workflow-Server zu einem anderen transferiert. Da Migrationen ebenfalls Kommunikationskosten verursachen, sollten sie nur dann eingesetzt werden, wenn die durch den günstigeren Workflow-Server erzielten Einsparungen größer sind als die zusätzlichen Kommunikationskosten. Um entscheiden zu können, welche Migrationen rentabel sind, wurden in dieser Arbeit Verteilungsalgorithmen und ein Kostenmodell entwickelt, durch die zur Modellierungszeit für jede Aktivität einer Workflow-Vorlage jeweils der (bzgl. der Gesamtkosten) günstigste Workflow-Server ermittelt wird. Dieser wird der entsprechenden Aktivität durch eine Serverzuordnung zugewiesen. Es gibt aber auch Szenarien, in denen diese statische Zuordnung von Aktivitäten zu Workflow-Servern nicht ausreichend ist. Dies ist immer dann der Fall, wenn die Menge der potentiellen Bearbeiter einer Aktivität von Vorgängeraktivitäten abhängt, weil dann zur Modellierungszeit noch nicht entschieden werden kann, welcher Workflow-Server für diese Aktivität am besten geeignet ist. Deshalb werden in dieser Arbeit die sog. variablen Serverzuordnungsausdrücke eingeführt, die zur Modellierungszeit automatisch berechnet werden können und eine effiziente Auswahl des Workflow-Servers zur Ausführungszeit der Workflow-Instanz ermöglichen.

Migrationen verursachen recht hohe Kommunikationskosten, insbesondere weil bei ihnen üblicherweise alle zu der betroffenen WF-Instanz vorhandenen Daten übertragen werden. In dieser Arbeit werden Verfahren entwickelt, bei denen nur die vom Zielserver einer Migration tatsächlich (noch) benötigten Daten übertragen werden. Da redundante Datenübertragung ausgeschlossen ist, wird das für Migrationen notwendige Datenvolumen minimiert. Durch die bisher erwähnten Maßnahmen kann die Belastung des Kommunikationssystems deutlich reduziert werden, so dass eine Überlastung der Netzwerkkomponenten verhindert wird. Jedoch kann die Belastung der Workflow-Server noch ein Problem darstellen. Dieses wird durch ein Verfahren gelöst, durch das mehrere Workflow-Server im selben Teilnetz betrieben werden können. Es erfordert keine zusätzliche Kommunikation und ermöglicht die Vorgabe einer beliebigen Lastaufteilung zwischen diesen Workflow-Servern.

ADEPT$_{distribution}$ wird im Rahmen der Diskussion qualitativ mit anderen aus der Literatur bekannten Verteilungsmodellen verglichen. Zusätzlich wird durch Simulationen auch noch quantitativ gezeigt, dass der vorgestellte Ansatz bzgl. der Belastung der Workflow-Server und des Kommunikationssystems am günstigsten ist. Außerdem wurden die wichtigsten der in dieser Arbeit vorgestellten Verfahren auf Basis eines kommerziellen Workflow-Produkts bzw. des ADEPT-Prototypen implementiert.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Teil I

Problemstellung und Grundlagen

Kapitel 1

Einleitung

Gegenstand dieser Arbeit ist die Entwicklung eines Workflow-Management-Systems (WfMS) für den unternehmensweiten Einsatz. Ein unternehmensweites Szenario zeichnet sich dadurch aus, dass sehr viele Benutzer mit dem WfMS arbeiten, was zu einer sehr großen von dem WfMS zu bewältigenden Last führt. Eine Forderung an das zu entwickelnde WfMS ist deshalb, dass es so konfigurierbar sein muss, dass keine Systemkomponente überlastet ist. Ist dies gegeben, so bezeichnen wir es als (bei wachsender Last) skalierbar.

Bei einem WfMS können nicht nur die Rechner, auf denen das WfMS läuft, überlastet sein, sondern auch weitere Systemkomponenten. So ist es ebenfalls möglich, dass das dem WfMS zugrunde liegende Kommunikationssystem überlastet ist, dass also bestimmte Teilnetze oder Gateways, welche die Teilnetze verbinden, überlastet sind. Dieser Aspekt wurde bisher in der im Forschungsgebiet der WfMS erschienenen Literatur nicht berücksichtigt. Deshalb ist es das Ziel dieser Arbeit, ein „effizientes WfMS“ zu entwickeln, bei dem eine möglichst geringe Belastung für das Kommunikationssystem entsteht. Doch auch der Aspekt einer potentiellen Überlastung einzelner Rechner, auf denen das WfMS läuft, wird nicht außer Acht gelassen: Die insgesamt zu bewältigende Last muss möglichst beliebig auf diese Rechner verteilt werden können, so dass eine derartige Überlastung ebenfalls verhindert werden kann.

Das vorliegende Kapitel ist wie folgt strukturiert: Abschnitt 1.1 bietet eine kurze Einführung in den Bereich der WfMS. Außerdem werden einige häufig verwendete Begriffe eingeführt. In Abschnitt 1.2 wird die in dieser Arbeit betrachtete Problemstellung ausführlich beschrieben und es wird belegt, dass die Belastung des Kommunikationssystems bei einem WfMS tatsächlich ein Problem darstellen kann. Schließlich wird in Abschnitt 1.3 der Aufbau der restlichen Arbeit erläutert.

1.1 Workflow-Management-Systeme

Im Folgenden wird erläutert, warum prozessorientierte Anwendungssysteme zunehmend an Bedeutung gewinnen. Außerdem wird aufgezeigt, dass solche Anwendungssysteme ausschließlich durch WfMS-Technologie mit vertretbarem Aufwand und zu akzeptablen Kosten realisiert werden können. Dann wird beschrieben, wie ein WfMS benutzt wird, wie es aufgebaut ist und welche Vorteile durch den Einsatz von WfMS resultieren. Schließlich wird noch diskutiert, warum Groupware-Systeme zur Realisierung von unternehmensweiten prozessorientierten Anwendungssystemen wenig geeignet sind.

1.1.1 Prozessorientierte Anwendungssysteme

In diesem Abschnitt wird geklärt, warum es sinnvoll ist, prozessorientierte Anwendungssysteme einzusetzen, und es wird untersucht, wie diese realisiert werden können.

1.1.1.1 Vorteile von prozessorientierten Anwendungssystemen

Heutzutage werden Arbeitsabläufe (engl. Workflows) üblicherweise arbeitsteilig ausgeführt. Das bedeutet, dass mehrere Personen jeweils eine Teilaufgabe bearbeiten. Da dabei für eine Person, die eine Einzelaufgabe bearbeitet, der Bezug zur Gesamtaufgabe oft nicht mehr erkennbar ist, wird diese Einzelaufgabe häufig zum Selbstzweck. Dies kann dazu führen, dass die Bearbeitung von Einzelaufgaben optimiert wird, ohne dass sich dadurch Vorteile für das Gesamtziel ergeben. Die Tendenz, dass Teilaufgaben unabhängig von der Gesamtaufgabe gesehen werden, wird durch die Dezentralisierung von Unternehmen und der verwendeten Rechnerwelt noch verstärkt.

Auch die zur Bearbeitung von Aufgaben eingesetzte Software orientiert sich an den Teilaufgaben. So bieten heutige Anwendungs- und Informationssysteme keine ausreichende Unterstützung zur Steuerung von Arbeitsabläufen (also zur Kontrolle der Gesamtaufgabe), die zu bearbeitenden Prozesse sind dem Informationssystem überhaupt nicht bekannt. Stattdessen erfolgt die elektronische Informationsverarbeitung i.d.R. funktions- und datenbezogen. Sie orientiert sich also an den Teilaufgaben. So unterstützen Anwendungssysteme üblicherweise einzelne Funktionen, wie die Abfrage von Kundendaten, die Erfassung eines Auftrags, das Erstellen eines Angebots, das Verbuchen eines Warenausgangs oder das Schreiben einer Rechnung. Die Beziehung zwischen diesen Einzelaufgaben ist dem Informationssystem nicht bekannt. Dass die Aufgaben einem gemeinsamen Arbeitsablauf angehören, ist nicht direkt ersichtlich. Der Zusammenhang zwischen den Teilaufgaben existiert nur implizit in den Anwendungsdaten, welche in einer von den Einzelfunktionen gemeinsam verwendeten Datenbank gespeichert sind. Die Information, in welcher Reihenfolge die Funktionen eines Arbeitsablaufs ausgeführt werden müssen, existiert nur in den Köpfen der prozessbeteiligten Personen (und evtl. in einem Organisationshandbuch). Die Einhaltung der Ausführungsreichenfolge wird vom Informationssystem nicht unterstützt. Deshalb muss relevante Vorgangsinformation durch die Benutzer explizit erfragt werden, anstatt dass sie vom Informationssystem automatisch bei der Ausführung einer Teilaufgabe zur Verfügung gestellt wird. Außerdem muss zum Starten des zur Bearbeitung einer Teilaufgabe notwendigen Programms zeitaufwendig durch Systemmenüs navigiert werden, anstatt dass dieses Programm vom Informationssystem automatisch gestartet wird. Schließlich besteht bei komplexen Arbeitsabläufen die Gefahr, dass Teilaufgaben vergessen, mehrfach ausgeführt oder in der falschen Reihenfolge bearbeitet werden. Dies lässt sich wegen der passiven Form der Informationsbereitstellung kaum verhindern.

Aus diesen Gründen besteht von Anwenderseite verstärkt der Wunsch nach Anwendungssystemen, welche sich an den Arbeitsabläufen orientieren. Um solche Anwendungssysteme realisieren zu können, müssen die elektronisch zu unterstützenden Arbeitsabläufe erkannt und modelliert werden (Geschäftsprozessmodellierung [Sch96, Sch98, VB96]). Bei dieser Gelegenheit macht es Sinn, die Abläufe zusätzlich noch zu analysieren und zu optimieren (Geschäftsprozess-Reengineering [Ham95]). Die (optimierten) Abläufe sollen durch ein Anwendungssystem umgesetzt werden, das selbstständig erkennt, welche Teilaufgaben (Prozessschritte) aktuell bearbeitet werden müssen. Eine solche Teilaufgabe soll den richtigen Personen zur Bearbeitung aktiv angeboten werden. Wählt eine der Personen die Aufgabe zur Bearbeitung aus, so soll das zugehörige Anwendungsprogramm mit

den richtigen Daten gestartet werden. Die Bearbeitung der Teilaufgaben ist dadurch wesentlich effizienter möglich. Dadurch, dass ein prozessorientiertes Anwendungssystem die Arbeitsabläufe nicht nur koordiniert, sondern zusätzlich auch noch überwacht, kann ein weit größeres Potential ausgeschöpft werden, als dies bei heutigen (funktionsorientierten) Informationssystemen möglich ist. So kann ein Benutzer frühzeitig auf drohende Terminverletzungen hingewiesen werden oder er kann ablaufbezogen informiert werden (z.B. über den Zustand des Ablaufs, über vorgenommene bzw. notwendige Ausnahmebehandlungen, etc.). Es steht aber außer Frage, dass vorgangsorientierte Informationssysteme dieselben Anforderungen erfüllen müssen, wie die heute eingesetzten funktionsorientierten Systeme. So muss mindestens derselbe Grad an Datenkonsistenz garantiert werden können und auch die Zuverlässigkeit und Verfügbarkeit dürfen nicht geringer sein. Außerdem muss sich das System bei unerwarteten Benutzereingaben, Fehler- und Ausnahmesituationen möglichst robust verhalten. Und natürlich dürfen die Antwortzeiten nicht größer sein, als bei den bisher eingesetzten Systemen, da den Anwendern keine längeren Wartezeiten zugemutet werden sollen. Das prozessorientierte Anwendungssystem muss also skalierbar sein, falls eine große Last zu bewältigen ist.

1.1.1.2 Realisierung von prozessorientierten Anwendungssystemen

Es existieren nun mehrere Möglichkeiten, wie prozessorientierte Anwendungssysteme realisiert werden können. Die naheliegendste Variante ist eine „konventionelle Realisierung“ direkt in den Anwendungsprogrammen, welche auch die Teilaufgaben realisieren. Dabei wird die Prozesslogik in Entscheidungstabellen oder If-Then-Else-Anweisungen „fest verdrahtet“. Diese Vorgehensweise ist allerdings nicht trivial und hat zudem zahlreiche Nachteile: So gehören i.d.R. nicht alle Funktionen eines Arbeitsablaufs zu demselben Anwendungssystem, sie werden nicht auf demselben Rechner bearbeitet und die benötigten Daten stammen nicht aus derselben Datenbank. Deshalb muss ein verteiltes Anwendungssystem realisiert werden, wozu systemnahe Kenntnisse erforderlich sind (z.B. aus den Bereichen Rechnerkommunikation oder Prozesssynchronisation). Über solche Kenntnisse verfügt ein Anwendungsentwickler normalerweise nicht. Um die oben beschriebenen Anforderungen nicht nur für einzelne Funktionen, sondern für gesamte Arbeitsabläufe erfüllen zu können, sind zusätzlich Kenntnisse aus den Bereichen Logging und Recovery erforderlich. Selbst wenn diese Kenntnisse vorhanden sind, so ist eine konventionelle Realisierung von prozessorientierten Anwendungssystemen zumindest sehr teuer und fehleranfällig. Doch der wohl bedeutendste Nachteil dieser Variante betrifft die Wartung prozessorientierter Anwendungssysteme. Während Programme, welche einzelne Teilaufgaben realisieren, sehr selten verändert werden müssen, ändern sich gesamte Anwendungsprozesse häufig. Schon eine Veränderung in der Ausführungsreihenfolge der Teilaufgaben führt dazu, dass das Anwendungssystem verändert werden muss. Da die Ablauflogik bei dieser Realisierungsvariante Bestandteil des Anwendungsprogramms ist, muss in dessen Programmcode eingegriffen werden, um die Änderung durchzuführen. Auch dies ist aufwendig und fehleranfällig. Weil solche Änderungen in prozessorientierten Anwendungssystemen häufig auftreten, scheidet die soeben vorgestellte Variante aus Kostengründen aus.

Alternativ kann ein prozessorientiertes Anwendungssystem, ebenso wie jedes andere verteilte Informationssystem, auf Basis einer verteilten Datenbank realisiert werden: Die Kommunikation findet (implizit) durch die in der verteilten Datenbank gespeicherten Datenobjekte statt. Dies hat den Vorteil, dass ortstransparent auf die Daten zugegriffen werden kann, da die Verteilung der Daten für die Anwendungsprogramme nicht sichtbar ist. Dadurch reduziert sich der Aufwand für die Erstellung des Anwendungssystems. Eine (verteilte) Datenbank ist aber ein passives System, so dass die Ablauflogik weiterhin in den Anwendungsprogrammen realisiert werden muss. Deshalb treten die im

vorherigen Absatz beschriebenen Probleme auch bei dieser Realisierungsvariante auf, so dass dieser Ansatz ebenfalls verworfen werden muss.

Nachdem wir einige Ansätze zur Realisierung von prozessorientierten Anwendungssystemen betrachtet haben, welche nicht zum gewünschten Resultat führen, wollen wir nun untersuchen, welche Anforderungen ein solches System zumindest erfüllen muss. Es lässt sich nur dann zuverlässig und zu vernünftigen Kosten realisieren, wenn es die folgenden Bedingungen erfüllt:

- Die Ablauflogik des Gesamtprozesses kann separat von den Prozessschritten (Anwendungsfunktionen) festgelegt werden. Zur besseren Strukturierung sollte es dabei möglich sein, den Kontroll- und den Datenfluss der Ablauflogik getrennt festzulegen.
- Die Implementierung der Anwendungsfunktionen sollte nicht wesentlich komplexer sein, als bei einer funktionsorientierten Realisierung des Anwendungssystems.
- Eine Veränderung der Aufrufreihenfolge der Anwendungsfunktionen darf sich nur in der Definition der Ablauflogik niederschlagen. Die Implementierung der Anwendungsfunktionen sollte davon nicht betroffen sein. Diese sind isolierte Anwendungskomponenten, die lediglich über wohl definierte Ein- und Ausgabeparameter mit dem Gesamtsystem kommunizieren. Dies macht es auch möglich, die Anwendungsfunktionen isoliert zu testen.
- Systemnahe Aspekte (z.B. Recovery, Logging, Inter-Prozess-Kommunikation) müssen von dem übergeordneten System zur Ablaufsteuerung behandelt werden und dürfen sich nicht in der Implementierung der Anwendungsfunktionen niederschlagen. Dadurch wird es möglich, dass Anwendungsfunktionen von „normalen Anwendungsentwicklern“ implementiert werden.

Diese Anforderungen lassen sich nur dann erreichen, wenn die Ablaufsteuerung von einem dafür spezialisierten System, einem sogenannten Workflow-Management-System, übernommen wird. Wie solche Systeme aufgebaut sind und welche Vorteile sie mit sich bringen, wird im nachfolgenden Abschnitt betrachtet.

1.1.2 Prozessorientierte Workflow-Management-Systeme

Der folgende Abschnitt bietet eine kurze Einführung in die Funktionsweise von WfMS (siehe auch [BS95, GHS95, Jab95, JBS97]) und eine Festlegung von in dieser Arbeit häufig verwendeten Begriffen. Das in der Arbeit verwendete Workflow-Modell wird hier nicht umfassend beschrieben, eine detaillierte Beschreibung dieses Modells wird in Kapitel 2 nachgeliefert. Im vorliegenden Abschnitt wird skizziert, wie Workflows (WF) definiert werden, wie sie ausgeführt werden, wie ein WfMS aufgebaut ist und welche Vorteile durch die Verwendung eines WfMS resultieren.

1.1.2.1 Anwendungsentwicklung mit Workflow-Management-Systemen

Wie schon erwähnt wurde, ist eine der essentiellen Eigenschaften von WfMS, dass die Festlegung eines Gesamtarbeitsablaufs getrennt von der Implementierung der Anwendungsfunktionen erfolgt. Die Phase, in der diese Festlegung stattfindet, wird als *Modellierungszeit* (engl. Buildtime) bezeichnet. Sie gliedert sich, wie in Abb. 1.1 dargestellt, in zwei Bereiche: Die Festlegung des Ablaufgraphen und die Implementierung der Anwendungsfunktionen. Der Ablaufgraph wird durch eine *WF-Vorlage* spezifiziert, welche einem *WF-Typ* zugeordnet und im WfMS hinterlegt wird. Die WF-Vorlage wird durch eine meist graphische Ablaufbeschreibungssprache spezifiziert und in ihr wird festgelegt, welche Teilschritte (*Aktivitäten*) in welcher Reihenfolge und unter welchen Bedingungen durchgeführt werden sollen (oberer Teil von Abb. 1.1). In der WF-Vorlage werden Aktivitäten durch Rechtecke,

Kreise oder andere Symbole und Bearbeitungsreihenfolgen zwischen ihnen durch Pfeile dargestellt. Es wird zudem spezifiziert, ob Aktivitäten sequentiell oder parallel ausgeführt werden sollen und ob es alternative Ausführungspfade (Verzweigungen) gibt. In einer WF-Vorlage wird außerdem der *Datenfluss* des WF-Typs festgelegt, d.h. es wird angegeben, welche Daten von welchen Aktivitäten gelesen bzw. geschrieben werden. Die Anwendungsfunktionen (unterer Teil von Abb. 1.1) sind den Aktivitäten zugeordnet und werden im folgenden auch als *Anwendungs-* oder *Aktivitätenprogramme* bezeichnet. Sie können unabhängig von der WF-Vorlage implementiert werden und natürlich auch an mehreren Stellen in einer WF-Vorlage oder in verschiedenen WF-Vorlagen verwendet werden. Auch eine Weiterverwendung von schon existierenden Programmbausteinen ist möglich.

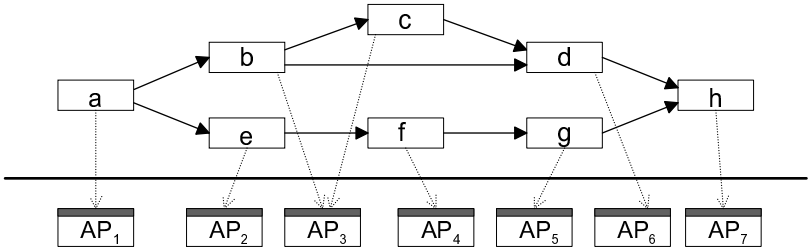


Abbildung 1.1 Grundprinzip der Trennung von Ablauflogik und Anwendungsprogrammen in WfMS.

Bei der Modellierung einer Prozessvorlage wird zudem festgelegt, von welchen Bearbeitern eine Aktivität ausgeführt werden soll. Dies erfolgt durch eine sogenannte *Bearbeiterzuordnung*. Es ist eher unüblich, dass einer Aktivität direkt ein Bearbeiter zugeordnet wird, da diese Vorgehensweise zu inflexibel ist. Stattdessen erfolgt die Bearbeiterzuordnung häufig durch eine *Rolle* (z.B. `Rolle = Arzt`), die auch den gewünschten Bearbeitern zugeordnet wird. Eine Rolle beschreibt bestimmte Fähigkeiten oder Kompetenzen, welche dem entsprechenden Benutzer zugeordnet sind. Die Zuordnung der Rollen zu den Benutzern wird in dem *Organisationsmodell* [RW94] des WfMS verwaltet. Bei der Festlegung einer Bearbeiterzuordnung kann in manchen Systemen auch eine *Organisationseinheit* (OE) spezifiziert werden, der die Bearbeiter angehören sollen, welche die entsprechende Aktivität bearbeiten dürfen (z.B. `Rolle = Arzt` $\wedge$ `OE = Radiologie`). Die Zuordnung von Personen zu Organisationseinheiten (z.B. Abteilungen) wird ebenfalls im Organisationsmodell verwaltet. Bei einigen WfMS können des Weiteren auch noch Vertreterregelungen festgelegt werden, die angeben, an wen eine Aufgabe weitergeleitet werden soll, falls die eigentlich dafür vorgesehene(n) Person(en) abwesend sind.

Um die Korrektheit eines modellierten WF verifizieren zu können, bieten manche WfMS die Möglichkeit zur Visualisierung bzw. Animation des Verhaltens einer WF-Vorlage [IBM96b]. Eine solche Validation ist auch dann möglich, wenn die Aktivitätenprogramme noch nicht implementiert wurden, da deren Ausführung nur simuliert wird (z.B. durch eine Maske, in der die Eingabedaten der Aktivität angezeigt werden und die Ausgabedaten angegeben werden können). Dies vereinfacht die Anwendungsentwicklung drastisch, weil WF-Vorlagen so zu einem sehr frühen Zeitpunkt getestet werden können.

Zusammenfassend lässt sich also feststellen, dass die wesentlichen Aufgaben zur Modellierungszeit eines WF-Typs die Beschreibung des Kontroll- und Datenflusses, die Implementierung und Zuordnung von Aktivitätenprogrammen und die Festlegung der Bearbeiter der Aktivitäten sind. Zusätzlich bieten einige Systeme noch weitere Möglichkeiten, wie die Festlegung von Vertreterregelungen oder die Einbeziehung von Zeitaspekten (z.B. Vorgabe von Maximaldauern von Aktivitäten).

1.1.2.2 Ausführung von Workflows

Wir wollen nun die Ausführung von Instanzen, der zur Modellierungszeit definierten WF-Typen, betrachten. Diese Phase wird als *Ausführungszeit* (engl. Runtime) bezeichnet. Um eine Instanz eines WF-Typs ausführen zu können, muss zuerst eine entsprechende *WF-Instanz* (kurz: Workflow) durch einen Benutzer des WfMS erzeugt werden. In der Regel wird dies durch Aufruf eines entsprechenden Aktivitätenprogramms, wie „Auftrag erfassen“ oder „Patient aufnehmen“ geschehen. Daraufhin steuert das WfMS die Bearbeitung der in diesem WF enthaltenen Aktivitäten. Steht dabei die Bearbeitung einer bestimmten Aktivität an, so erzeugt das WfMS eine *Aktivitäteninstanz* vom entsprechenden Aktivitätentyp. Wir werden Aktivitäteninstanzen im Folgenden auch kurz als Aktivitäten bezeichnen, falls eine Verwechslung mit den Aktivitätentypen ausgeschlossen ist.

Wenn nun eine Aktivitäteninstanz zur Bearbeitung ansteht, so ermittelt das WfMS aus der Bearbeiterzuordnung des zugehörigen Aktivitätentyps die Benutzer, welche diese Aktivitäteninstanz bearbeiten dürfen. Die Bearbeiterzuordnung kann vorsehen, dass die potentiellen Bearbeiter der Aktivität eine bestimmte Rolle innehaben oder einer bestimmten OE angehören müssen. In diesen Fällen erfordert die Bearbeiterauflösung einen Zugriff auf das Organisationsmodell, um die geeigneten Bearbeiter einer Aktivitäteninstanz zu ermitteln. Nachdem diese ermittelt wurden, wird die Aktivitäteninstanz in deren *Arbeitslisten* eingetragen. Wählt später einer dieser Benutzer diese Aktivität aus, so wird er zum Bearbeiter der Aktivitäteninstanz. Der entsprechende Eintrag wird aus den Arbeitslisten der anderen potentiellen Bearbeiter entfernt. Das WfMS synchronisiert die Auswahl von Arbeitslisteneinträgen, so dass jede Aktivitäteninstanz nur einem Bearbeiter zugeordnet wird. Auf dem Rechner dieses Bearbeiters wird das zu der Aktivitäteninstanz gehörende Aktivitätenprogramm gestartet und mit den zugehörigen Daten versorgt. Wenn der Benutzer die Bearbeitung der Aktivitäteninstanz beendet hat, so werden die Ausgabedaten zum WfMS transportiert, so dass dieses zur nächsten Aktivität weiterschalten kann, für die dann dieselbe Prozedur durchgeführt wird.

Häufig erlauben es WfMS auch, *automatische Aktivitäten* zu definieren, die ohne Benutzerinteraktion bearbeitet werden. Eine solche Aktivität kann z.B. verwendet werden, um einen Zugriff auf Daten einer Datenbank (DB) zur realisieren, oder um eine Berechnung durchzuführen. Bei der Ausführung einer Instanz einer automatischen Aktivität entfällt die Auflösung der Bearbeiterzuordnung und die Aktivität wird auch in keine Arbeitslisten eingetragen. Stattdessen wird das zugehörige Anwendungsprogramm automatisch gestartet, sobald die Aktivität zur Bearbeitung ansteht.

1.1.2.3 Aufbau von Workflow-Management-Systemen

WfMS bestehen i.d.R. aus zwei Hauptkomponenten: WF-Server und WF-Client. Der Aufbau eines WfMS ist in Abb. 1.2 stark vereinfacht dargestellt, eine detaillierte Beschreibung folgt in Abschnitt 3.1. Ein *WF-Server* (kurz Server) steuert die Ausführung der WF-Instanzen. Dabei speichert er Information über die von ihm kontrollierten WF-Instanzen in seiner *WF-Datenbank* (WF-DB). Diese enthält zusätzlich Information zu den WF-Typen, zum Organisationsmodell und Metainformation über das WfMS. Ein WF-Server ist über ein Kommunikationsnetzwerk mit den *WF-Clients* verbunden. Diese stellen die Schnittstelle zu den Benutzern des WfMS dar. Sie zeigen die Arbeitslisten der Benutzer an, ermöglichen ihnen die Auswahl von Aktivitäteninstanzen und sorgen dafür, dass die Aktivitätenprogramme korrekt gestartet werden. Um diese Aufgaben erfüllen zu können, ist Kommunikation zwischen WF-Server und WF-Client notwendig. Zwischen diesen Systemkomponenten muss sogar sehr viel kommuniziert werden, weil bei jeder Aktualisierung einer Arbeitsliste eines Benutzers und beim Starten und Beenden jedes Teilschrittprogramms kommuniziert werden

muss. Insbesondere letzteres erfordert evtl. die Übertragung von großen Datenmengen, da die in der WF-DB des WF-Servers gespeicherten (möglicherweise sehr großen) Parameterdaten über das Kommunikationssystem vom/zum WF-Client übertragen werden müssen. Eine Alternative zur Verwaltung von Anwendungsdaten in der WF-DB stellt deren Speicherung in einer *externen Datenquelle* dar. In welchen Fällen dies sinnvoll ist, wird in Abschnitt 2.3.1 noch ausführlich erörtert. Doch auch bei einer Speicherung von Parameterdaten in einer externen Datenquelle, müssen die zur Ausführung eines Teilschrittprogramms benötigten Daten zum WF-Client übertragen werden. Bezüglich des Kommunikationsaufwandes ist diese Alternative also nicht von Vorteil.

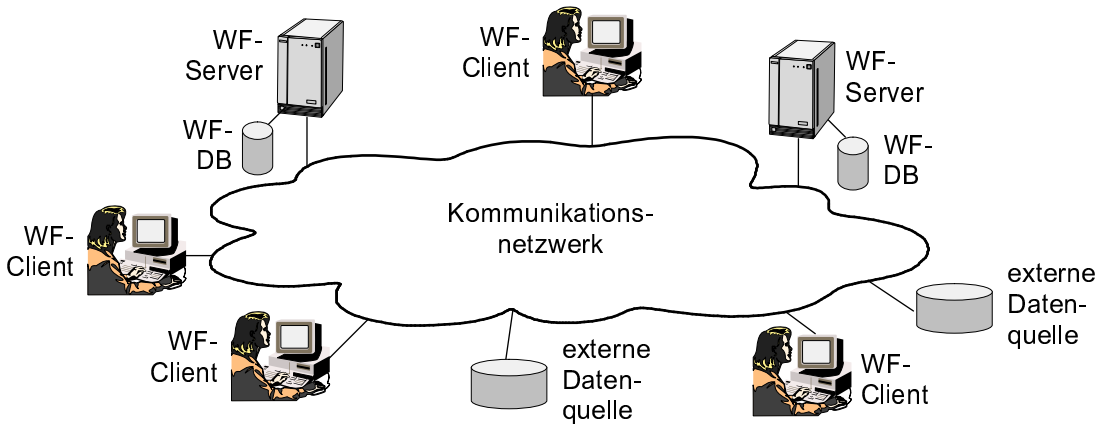


Abbildung 1.2 Aufbau eines WfMS (stark vereinfacht).

Um die Leistungsfähigkeit und Verfügbarkeit eines WfMS zu steigern, werden häufig Systeme mit mehreren WF-Servern verwendet (siehe Abb. 1.2). Solche bezeichnen wir als *verteilte WfMS*. Die WF-Server bilden dann gemeinsam die *WF-Engine*. Prinzipiell kann in einem verteilten WfMS jede Aktivitäteninstanz von einem beliebigen WF-Server kontrolliert werden. Welcher Server eine Aktivitäteninstanz tatsächlich kontrollieren soll, wird durch die *Serverzuordnung* des zugehörigen Aktivitätentyps bestimmt. Nach welchen Kriterien eine solche Serverzuordnung festgelegt werden soll, wird in dieser Arbeit noch ausführlich untersucht. Mit Hilfe einer Serverzuordnung kann der WF-Server, welcher die Instanzen eines Aktivitätentyps kontrollieren soll, fest vorgegeben werden. Solche Serverzuordnungen bezeichnen wir im Folgenden als *statische Serverzuordnungen*. Es ist aber auch möglich, als Serverzuordnung einen Ausdruck anzugeben, der vor der Ausführung einer Aktivitäteninstanz ausgewertet wird, um denjenigen WF-Server zu ermitteln, der diese Aktivitäteninstanz kontrollieren soll. Eine solche Serverzuordnung bezeichnen wir als *variable Serverzuordnung*. Welche Ausdrücke als variable Serverzuordnungen verwendet werden können und welche Vorteile die Verwendung solcher Ausdrücke mit sich bringt, wird in Kapitel 5 noch ausführlich untersucht. Es ist nun möglich, die Serverzuordnungen so festzulegen, dass verschiedene Aktivitäteninstanzen einer WF-Instanz von unterschiedlichen Servern kontrolliert werden. Wenn zwei aufeinanderfolgende Aktivitäteninstanzen von verschiedenen Servern kontrolliert werden, so wird zwischen ihnen eine *Migration* der WF-Instanz notwendig. Das bedeutet, bevor die Kontrolle über die WF-Instanz vom *Quellserver* der Migration an den *Zielserver* übergeben werden kann, müssen die für diese WF-Instanz relevanten Daten aus der WF-DB des Quellservers an den Zielserver übertragen werden. Natürlich belastet auch eine solche Migration das Kommunikationssystem.

1.1.2.4 Vorteile von Workflow-Management-Systemen

Wir haben schon festgestellt, dass die Verwendung eines WfMS die einzige Möglichkeit ist, um ein prozessorientiertes Anwendungssystem mit vertretbarem Aufwand und zu akzeptablen Kosten zu realisieren. Der nun folgende Abschnitt fasst die Vorteile, die WfMS bieten, nochmals zusammen. Dabei handelt es sich um Vorteile, welche sich dem Anwendungsentwickler bieten, und um solche, von denen der Anwender profitiert. Da natürlich alles, was mit einem WfMS möglich ist, auch durch eine konventionelle Implementierung realisiert werden kann (mit entsprechend höherem Aufwand), lassen sich die Vorteile für den Anwender allerdings auch auf reduzierten Aufwand bei der Anwendungsentwicklung zurückführen. Im Einzelnen sind als Vorteile von WfMS zu nennen:

- Geschäftsprozesse lassen sich i.d.R. direkt auf das verwendete WF-Modell abbilden, sofern alle benötigten Konstrukte wie Parallelität, Verzweigungen und Schleifen angeboten werden. Zudem ist dieser Vorgang überschaubarer als bei konventioneller Implementierungstechnik, weil von der internen Ablauflogik der Teilschrittprogramme abstrahiert wird. Dadurch lassen sich Umsetzungsfehler vermeiden oder zumindest deutlich reduzieren.
- Die Existenz einer formalen Beschreibung der Ablauflogik (WF-Vorlage) ermöglicht es, das zukünftige Systemverhalten schon zur Modellierungszeit zu überprüfen. Durch Animationen kann evaluiert werden, ob ein modellierter Ablauf dem Soll-Prozess entspricht. Hierdurch können Entwurfsfehler frühzeitig erkannt und korrigiert werden.
- Der wohl größte Vorteil von WfMS wird deutlich, wenn ein Geschäftsprozess verändert werden muss. So kann die Ausführungsreihenfolge der Aktivitäten einer WF-Vorlage normalerweise verändert werden, ohne dass die Teilschrittprogramme davon betroffen sind. Dagegen muss bei einem konventionell realisierten Anwendungssystem bei Modifikationen stets der Programmcode verändert werden. Bei Verwendung eines WfMS sind allenfalls sehr keine Anpassungen von Anwendungsprogrammen notwendig, weshalb die Wartungskosten erheblich reduziert werden.
- Ein WfMS bietet gewöhnlich zahlreiche Systemdienste, z.B. für Fehler- und Ausnahmeüberwachung, Wiederanlauf nach einem Systemzusammenbruch, Überwachung von Zeitvorgaben, etc. Durch deren Nutzung kann der für die Entwicklung des Anwendungssystems nötige Aufwand deutlich reduziert werden.
- WfMS verfügen über Systemschnittstellen zur Verwaltung der Arbeitslisten der Benutzer. Häufig können auch unterschiedliche Sichten auf eine Arbeitsliste eingerichtet werden, z.B. um Tätigkeiten nach Typen oder Prioritäten zu gruppieren. Wird der vom WfMS angebotene „Standard-Client“ verwendet, so entfällt der für die Entwicklung von Benutzerschnittstellen notwendige Aufwand völlig. Ist dessen Funktionalität nicht ausreichend, so dass eigene Programme z.B. zur Visualisierung der Arbeitslisten entwickelt werden müssen, so können zumindest die angebotenen Systemdienste verwendet werden, um deren Entwicklung zu vereinfachen.
- Nachdem ein Benutzer eine Aufgabe aus seiner Arbeitsliste ausgewählt hat, übernimmt das WfMS das Starten des benötigten Anwendungsprogramms und dessen Datenversorgung, so dass für den Benutzer das Navigieren durch Systemmenüs entfällt.
- Da ein WfMS die gesamten Arbeitsabläufe kennt (anstatt funktionsorientiert zu arbeiten) können die Abläufe aktiv überwacht werden. Dadurch ist es möglich, die Benutzer vor entstehenden Problemen zu warnen. So können z.B. Terminüberschreitungen vor ihrer Entstehung erkannt und somit verhindert werden. Außerdem kann von einem (dazu berechtigten) Benutzer vom WfMS Information zu einem Ablauf eingeholt werden. So ist es z.B. möglich, den Bearbeitungszustand einer laufenden WF-Instanz zu ermitteln, oder auch die Bearbeitung einer schon abgeschlossenen WF-Instanz nachzuvollziehen.

- Die bei der Durchführung einer WF-Instanz ausgeführten Aktionen können vom WfMS protokolliert und diesem WF zugeordnet werden. Insbesondere lassen sich so die Bearbeitungszeiten von Aktivitäteninstanzen und Prozessen ermitteln. Diese Daten können dann z.B. zu einer prozessorientierten Kostenrechnung verwendet werden.

1.1.3 Abgrenzung gegenüber Groupware-Systemen

Das Ziel von *Groupware*-Systemen ist es, Teamarbeit in i.d.R. kleinen Arbeitsgruppen zu unterstützen. Während ein WfMS einen Arbeitsprozess steuert und die Benutzer dabei getrennt betrachtet, steht bei einem Groupware-System deren Zusammenarbeit im Vordergrund. Als Groupware wird die Technologie bezeichnet, welche das zum Erreichen eines gemeinsamen Ziels notwendige „Teamwork“ unterstützt, während das zugehörige Forschungsgebiet den Namen *Computer Supported Cooperative Work* (CSCW) trägt.

Groupware-Systeme lassen sich danach klassifizieren, ob die Mitglieder der Gruppe, deren Zusammenarbeit das Groupware-System unterstützt, sich am selben Ort befinden oder an unterschiedlichen Orten arbeiten, und außerdem, ob sie zur gleichen Zeit aktiv sind (synchron) oder zu unterschiedlichen Zeiten (asynchron) [EGR91, LK91]. Aufgrund dieser beiden Kriterien ergeben sich vier Kategorien, in welche Groupware-Systeme eingeordnet werden können. Um ein besseres Verständnis von dem Begriff Groupware zu erhalten, wollen wir nun einige konkrete Beispiele für Groupware-Systeme betrachten (siehe auch [EGR91]):

- *Message-Systeme* dienen zum elektronischen (asynchronen) Austausch von Nachrichten. Beispiele sind E-Mail-Systeme oder elektronische Bulletin-Board-Systeme.
- *Multi-User-Editoren* ermöglichen das gemeinsame Bearbeiten eines Dokuments durch mehrere Autoren (Joint-Editing).
- *Group-Decision-Support-Systeme* [KK88] unterstützen das gemeinsame Fällen einer Entscheidung durch mehrere Personen. Diese Systeme verwenden häufig die Metapher eines elektronischen Besprechungsraums.
- Unter dem Oberbegriff *Computer Conferencing* werden Systeme zusammengefasst, bei denen der Computer primär als Kommunikationsmedium dient. Ein Beispiel hierfür ist eine Videokonferenz, bei welcher Bild und Ton der Teilnehmer an die anderen Teilnehmer übertragen werden. Bei einer Desktop-Konferenz können die Teilnehmer zusätzlich auch noch gemeinsam Anwendungen ausführen.

Wie man an den Beispielen erkennen kann, sind Kommunikation und Kooperation wesentliche Aspekte eines Groupware-Systems. Um die Zusammenarbeit einer Gruppe zu unterstützen, ist aber auch Koordination notwendig. Da es sich bei der Koordination um die Hauptaufgabe jedes WfMS handelt, kann die Fähigkeit eines Groupware-Systems zur Koordination genutzt werden, um einen Arbeitsablauf zu steuern. Wie eine Ablaufsteuerung mit Lotus Notes, dem Marktführer unter den Groupware-Systemen, realisiert werden kann, wird in [DS97] beschrieben; in [HK95] findet sich eine Fallstudie für die Realisierung einer WF-Anwendung auf Basis von Lotus Notes. Prinzipiell existieren die folgenden beiden Möglichkeiten, um mit Lotus Notes die Ausführung von WF-Instanzen (Arbeitsabläufen) zu steuern:

- Bei einer E-Mail-basierten Realisierung der WF-Steuerung wird eine WF-Instanz nach der Bearbeitung einer Aktivitäteninstanz an den Bearbeiter des nächsten Teilschritts geschickt. Ein Nachteil dieser Vorgehensweise ist, dass die WF-Instanz nicht von Dritten eingesehen werden kann, da sie sich ausschließlich in dem Eingangskorb des E-Mail-Empfängers befindet. Deshalb ist

es fast unmöglich, den aktuellen Bearbeitungszustand einer WF-Instanz zu ermitteln. Außerdem kann eine Aktivität kaum mehreren potentiellen Bearbeitern zugeordnet werden. Dann müsste die WF-Instanz nämlich an mehrere Nachfolger geschickt werden, die sich vor der Bearbeitung des Arbeitsschritts auch noch synchronisieren müssten.

- Die Ablaufsteuerung kann auch auf Basis der (verteilten) Groupware-DB realisiert werden, d.h. alle WF-Instanzen werden in dieser DB gespeichert. In einem Steuerfeld der WF-Instanz wird dann der jeweils nächste Bearbeiter eingetragen, so dass die WF-Instanz in dessen – entsprechend konfigurierter – Ansicht (Arbeitsliste) erscheint. Bei dieser Realisierungsvariante kann auf die von der Groupware-DB verwalteten Daten der WF-Instanz von allen (berechtigten) Benutzern zugegriffen werden, so dass die bei der E-Mail-basierten Realisierung beschrieben Einschränkungen nicht gelten.

Bei diesen Realisierungsvarianten ist zu beachten, dass eine WF-Instanz manuell durch einen Benutzer an den Bearbeiter der Nachfolgeraktivität gesendet werden muss, oder dass dieser Bearbeiter manuell in ein Steuerfeld eingetragen werden muss. Die Ablaufsteuerung erfolgt also explizit durch die Benutzer. Dies ist zwar bei den für Groupware-Anwendungen typischen unstrukturierten Abläufen wünschenswert, für eine unternehmensweite Ablaufsteuerung aber nicht akzeptabel. Abhilfe kann hier geschaffen werden, indem die potentiellen Bearbeiter der Nachfolgeraktivität nicht manuell vom Benutzer festgelegt, sondern automatisch berechnet werden. Dies lässt sich z.B. bei Lotus Notes dadurch realisieren, dass LotusScript-Programme erstellt und bestimmten Schaltflächen („Buttons“) zugeordnet werden. Möchte ein Benutzer einen Arbeitsschritt beenden, so aktiviert er die entsprechende Schaltfläche, woraufhin das zugehörige LotusScript-Programm ausgeführt wird. Dieses berechnet dann die potentiellen Bearbeiter der Nachfolgeraktivität. Durch diese Vorgehensweise kann ein Ablauf fest vorgegeben werden. Er wird aber nicht (z.B. graphisch) modelliert, sondern er ergibt sich durch die Programmlogik zahlreicher LotusScript-Programme.

Wir wollen nun die Auswirkungen der soeben vorgeschlagenen Vorgehensweise analysieren. Groupware-Systeme verfügen über eine wenig ausgeprägte Entwicklungsumgebung zur Realisierung von WF-Anwendungen [HK95], was deren Implementierung wesentlich erschwert. Solche Entwicklungsumgebungen werden bei Groupware-Systemen auch nicht für notwendig erachtet, da die Systeme für die Unterstützung von kleinen Gruppen konzipiert sind, und nicht für den unternehmensweiten Einsatz. Das Fehlen einer ausgereiften Entwicklungsumgebung führt aber dazu, dass die WF-Vorlagen nicht explizit modelliert werden können, sondern, dass die Ablauflogik durch viele kleine Programme (vgl. LotusScript) realisiert werden muss. Dies führt wiederum zu einigen Nachteilen, wenn dennoch unternehmensweite prozessorientierte Anwendungssysteme auf Basis eines Groupware-Produkts realisiert werden:

- Da keine explizite Ablaufbeschreibung existiert, ist eine Validierung des Ablaufs durch eine Animation oder Simulation der WF-Vorlage nicht möglich. Da die Prozesse deshalb kaum ausreichend getestet werden können, ist die für ihren unternehmensweiten Einsatz erforderliche Zuverlässigkeit nicht zu erreichen.
- Die Performanz einer Groupware-DB ist deutlich schlechter als die einer konventionellen DB, da bei Groupware-Systemen eine wesentlich größere Flexibilität benötigt wird (z.B. Speicherung von ganzen Dokumenten in einem Feld) [DS97]. Deshalb kann bei einer Realisierung eines prozessorientierten Anwendungssystems auf Basis einer Groupware-DB, die für den unternehmensweiten Einsatz benötigte Skalierbarkeit nicht erreicht werden. Die andere Alternative, die Realisierung des Anwendungssystems auf Basis des E-Mail-Dienstes eines Groupware-Systems, scheidet aus den oben genannten Gründen i.d.R. aus.

- Da keine WF-Vorlage zur Ausführung der WF-Instanzen existiert, muss der zu realisierende WF von dem Anwendungsentwickler manuell in entsprechende (LotusScript-) Programme umgesetzt werden. Dies erfordert einen sehr hohen Aufwand bei der Anwendungsentwicklung und ist zudem sehr fehleranfällig.
- Der Programmcode, durch den die WF-Steuerung realisiert wird, ist auf sehr viele Felder zahlreicher Dokumente verteilt. Dadurch entsteht ein sehr großer Pflege- und Wartungsaufwand. Insbesondere die Veränderung der Ausführungsreihenfolge von Aktivitäten eines Ablaufs wird dadurch extrem aufwendig und fehleranfällig.
- Da alle Information über einen Ablauf in Programmfragmenten steckt, ist zudem nur sehr wenig (nutzbare) Information über die Struktur eines Prozesses verfügbar. So ist es z.B. sehr schwierig, die maximale Ausführungszeit eines Prozesses abzuschätzen, da nicht einmal die Nachfolger einer Aktivität explizit bekannt sind. Auf Basis eines solchen Systems wäre es fast unmöglich, die in dieser Arbeit präsentierten Verfahren zu realisieren. Dieses Problem wird durch die Existenz eines (graphischen) Aufsatzes zur Modellierung der Prozesse (wie er z.B. für Lotus Notes angeboten wird) nicht gelöst. Vielmehr benötigt die WF-Engine zur Ausführungszeit explizite Information über die Struktur der WF-Instanzen.

Da keine WF-Vorlagen existieren, lassen sich also die beim unternehmensweiten Einsatz entstehenden Anforderung an Leistungsfähigkeit, Zuverlässigkeit und Änderbarkeit eines Anwendungssystems bei einer auf einem Groupware-System basierenden Realisierung nicht erfüllen. Außerdem fehlt Groupware-Systemen gegenüber WfMS viel Funktionalität, welche bei der Realisierung von prozessorientierten Anwendungen hilfreich ist [DS97]. Deshalb müssen komplexe Verfahren z.B. zur Bearbeiterzuordnung oder zur Zeitüberwachung vom Anwendungsentwickler selbst realisiert werden, was einen sehr großen Zusatzaufwand bedeutet. Zusammenfassend lässt sich also feststellen, dass ein Groupware-System eben kein WfMS ist und auch keine geeignete Basis zur Realisierung von prozessorientierten Anwendungssystemen für den unternehmensweiten Einsatz bietet. Deshalb wird diese Vorgehensweise auch nicht weiter verfolgt. Wenn wir im Folgenden von WfMS sprechen, so sind also stets prozessorientierte WfMS (wie in Abschnitt 1.1.2 beschrieben) gemeint.

1.2 Motivation für diese Arbeit

Die heute verfügbaren WfMS befinden sich noch in einem sehr frühen Entwicklungsstadium, das etwa mit der relationalen DB-Technologie zu Beginn der 80-er Jahre verglichen werden kann. Deshalb erfüllen sie bei weitem noch nicht alle Anforderungen, die an sie gestellt werden. Welche dies im Detail sind, ist z.B. in [DRK00, GHS95] aufgeführt. Um die Unzulänglichkeiten heutiger WfMS zu überwinden, wurden im Rahmen des ADEPT[1]-Projektes [DKR+95, DR98, RBD00] u.a. die folgenden Aspekte betrachtet: die dynamische Änderung der Struktur von WF-Instanzen zu deren Ausführungszeit [RD98, Rei00], die Überwachung von Zeitbedingungen zwischen den Aktivitäten eines Prozesses [Gri97], die Auswirkungen von Abhängigkeiten zwischen Aktivitäten unterschiedlicher WF-Instanzen [Hei00, Hei01] und das (partielle) Zurücksetzen von WF-Instanzen [Rei00]. Auf diese fortschrittlichen WF-Konzepte wollen wir nicht vertieft eingehen. Die in dieser Arbeit entwickelten Verfahren müssen aber so gestaltet sein, dass die Realisierung dieser WF-Konzepte nicht beeinträchtigt wird. Deshalb werden einige dieser Konzepte an geeigneter Stelle diskutiert.

[1] ADEPT steht für Application Development Based on Encapsulated Pre-Modeled Process Templates.

In der vorliegenden Arbeit wollen wir uns auf die Entwicklung eines skalierbaren WfMS konzentrieren. Die Bedeutung dieser Formulierung wird im nachfolgenden Abschnitt noch genauer erläutert. Anschließend wird an einem konkreten Beispiel gezeigt, dass es sich bei der untersuchten Fragestellung tatsächlich um ein sehr brisantes Problem handelt. Schließlich werden noch die Rahmenbedingungen geklärt, unter denen der in dieser Arbeit vorgestellte Ansatz verwendet werden kann.

1.2.1 Problemstellung

In der vorliegenden Arbeit wird untersucht, wie sehr große prozessorientierte Anwendungssysteme realisiert werden können. Damit sind Informationssysteme mit hunderten oder tausenden von Benutzern gemeint. Das einem solchen Anwendungssystem zugrundliegende WfMS muss viele tausend WF-Instanzen zur gleichen Zeit verwalten können. Es ist durchaus realistisch anzunehmen, dass prozessorientierte Anwendungssysteme mit bis zu 20000 Benutzern benötigt werden. So werden in [KAGM96, SK97] WF-basierte Anwendungen beschrieben, bei denen die Zahl der Benutzer bis auf einige zehntausend anwachsen kann oder mehrere zehntausend WF-Instanzen gleichzeitig im System sein können. Solche Zahlen können bei unternehmensweiten Anwendungen in Großkonzernen leicht erreicht werden. Auch für unternehmensübergreifende Anwendungssysteme sind diese Größenordnungen denkbar. Allerdings befinden sich dabei aufgeworfene weitere Fragestellungen, die unabhängig von der Skalierbarkeit sind, nicht im Fokus dieser Arbeit.

Doch nicht nur durch die hohen Benutzerzahlen wird eine große Last erzeugt. Wir betrachten Anwendungssysteme, bei denen außerdem von den Benutzern sehr große Datenmengen benötigt werden. Es sollen also realistische Datenvolumina (z.B. für Multimedia-Dokumente, CAD-Zeichnungen oder eingescannte Dokumente) unterstellt werden und nicht nur „Spieldaten“. Außerdem kann das zu realisierende Anwendungssystem weiträumig verteilt sein. Diese Annahme ist durchaus realistisch, da Großkonzerne in aller Regel weltweit über Niederlassungen oder Außenstellen verfügen. Zumindest ist aber anzunehmen, dass diese über eine sehr große Region (z.B. gesamte USA) verteilt sind. Deshalb muss die notwendige Kommunikation über ein Wide Area Network (WAN) abgewickelt werden. In einem solchen Szenario muss das WfMS nicht nur sehr viele Aktionen (z.B. Arbeitsliste aktualisieren, Aktivitäteninstanz startet/beenden) ausführen, sondern es muss zusätzlich eine extrem große Datenmenge (über ein WAN) übertragen werden.

Unter diesen Voraussetzungen ist klar, dass die hohe Belastung der WF-Engine ein Problem darstellt. Dies wurde auch schon erkannt und ist allgemein anerkannt, weshalb sich in der Literatur zahlreiche Vorschläge für Multi-Server-Architekturen für WfMS finden (z.B. in [AKA+94, CGS97, DKM+97, MWW+98, SNS99]). Die Vermeidung der Überlastung von WF-Servern ist auch eine der Problemstellungen der vorliegenden Arbeit. Zusätzlich stellt aber auch die Belastung des dem WfMS zugrunde liegenden Kommunikationssystems ein Problem dar, weil bei jeder auszuführenden Aktion eine Kommunikation zwischen einem WF-Server und einem Client stattfindet. Da jede dieser zahlreichen Kommunikationen auch noch eine sehr große Datenmenge umfassen kann (z.B. für Ein- und Ausgabeparameter von Aktivitätenprogrammen), muss das Kommunikationssystem zusätzlich zu der hohen Nachrichtenanzahl auch noch ein sehr großes Datenvolumen bewältigen. Deshalb müssen die in dieser Arbeit entwickelten Verfahren in der Lage sein, die Belastung für das Kommunikationssystem zu reduzieren. Dies ist insbesondere bei weiträumig verteilten Anwendungssystemen wichtig, da Kommunikation über ein WAN naturgemäß relativ langsam und teuer ist. Die gesamte Problematik der von WfMS erzeugten Kommunikationslast wurde bisher in der WF-Literatur noch nicht berücksichtigt.

Man könnte nun als Gegenargument anführen, dass Hochleistungskommunikationsnetzwerke [Par94] wie z.B. ATM (Asynchronous Transfer Mode [BM93, Onv92]) ausreichend schnell sind, um die anfallende Kommunikationslast zu bewältigen, und außerdem in Zukunft noch schneller werden. Deshalb könnte man die von WfMS erzeugte Kommunikationslast einfach ignorieren, wenn man ein ausreichend leistungsfähiges Kommunikationssystem verwendet. Diese „Strategie“ führt aber aus den folgenden Gründen zu keiner dauerhaften, allgemeingültigen und praktikablen Lösung des Problems:

- Ein Hochleistungs-WAN mit ausreichender Bandbreite ist sehr teuer. Diese Kosten sind unnötig, da die kommunizierte Datenmenge durch ein intelligent gestaltetes WfMS reduziert werden kann. Deshalb sollten diese Kosten vermieden werden. Außerdem sind Hochleistungsnetzwerke häufig nicht optimal für Übertragungen mit stark schwankender Datenrate geeignet [SVBC91, GTMG91], welche bei WfMS aber die Regel sind. So findet z.B. zu dem Zeitpunkt viel Kommunikation statt, in dem eine Aktivitäteninstanz gestartet wird und die dafür notwendigen großen Parameterdaten übertragen werden müssen. Dies führt zu der stark schwankenden Datenrate. Systeme wie ATM, bei denen eine bestimmte Bandbreite reserviert (und bezahlt) werden muss, sind aber eher für Übertragungen (z.B. von Videos) mit konstanter Datenrate geeignet.
- Werden in einem Anwendungssystem extrem große Daten verwendet (z.B. Multimediadaten mit einem hohen Anteil an Videoinformation), so kann man an die Grenzen der zur Verfügung stehenden Kommunikationstechnologie stoßen. Dies führt zu langen Wartezeiten für die Benutzer des Systems. Für die Zukunft ist zwar die Entwicklung von noch schnelleren Kommunikationsmedien zu erwarten, es ist aber durchaus möglich, dass diese Leistungssteigerung durch den Wunsch der Anwender nach Computerunterstützung für immer anspruchsvollere Anwendungen aufgezehrt wird. So kann man sich z.B. für den Krankenhausbereich vorstellen, dass zukünftig alle Daten inklusive hochauflösender Röntgenbilder und Computertomographie-Filmen elektronisch verfügbar und durch ein WfMS zugreifbar gemacht werden sollen.
- Hochleistungskommunikationssysteme bieten eine sehr große Bandbreite für die Datenübertragung. Sie haben aber auch ihre Schwachpunkte, wie z.B. die für einen Verbindungsaufbau benötigte Zeit, die Belastung durch eine große Anzahl von (kleinen) Nachrichten oder die Latenzzeit beim Zugang zum Kommunikationsmedium. Deswegen kann es selbst bei einem bzgl. der Übertragungsrate nicht überlasteten Kommunikationssystem, bei einer weit entfernten Kommunikation zwischen Benutzer und WF-Server, zu inakzeptablen Verzögerungen kommen. Schon um diese zu vermeiden, sollte auf unnötige WAN-Kommunikation verzichtet werden.
- Viele Anwender möchten kein Hochleistungs-WAN einsetzen, sondern preiswert z.B. über das Internet kommunizieren. So gibt es zahlreiche WfMS, welche die Kommunikation zwischen WF-Server und Client ausschließlich über das Internet abwickeln. In [MSKP97] werden einige kommerzielle WfMS dieses Typs vorgestellt. Aber auch Forschungsprototypen wie Panta Rhei [EGL96a, GE96], WASA [VWW96a], WebFlow [GMPP97] und METEOR$_2$ WebWork [MPS$^+$98] haben sich für diese Art der Kommunikation entschieden.

Das Ziel des in dieser Arbeit entwickelten Ansatzes ADEPT$_{distribution}$ (im Folgenden auch kurz ADEPT genannt, wenn eine Verwechslung mit dem Gesamtprojekt ausgeschlossen ist) ist es aber nicht nur, ein Konzept für ein WfMS zu entwickeln, durch das die Belastung für die WF-Server und für das Kommunikationssystem reduziert wird. Das zu entwickelnde System soll außerdem die Anforderungen erfüllen, die realistischerweise in der Praxis an ein WfMS gestellt werden. Ein Aspekt, dem – aufgrund der in dieser Arbeit gewählten Vorgehensweise – dabei besondere Beachtung geschenkt werden muss, sind die Bearbeiterzuordnungen. In der Praxis ist es sicherlich nicht ausreichend, wenn die potentiellen Bearbeiter einer Aktivität durch ihre Rolle und evtl. noch durch die Zugehörigkeit zu einer OE spezifiziert werden können. Häufig muss es möglich sein, dass die potentiellen Bearbeiter

einer Aktivität abhängig von den Bearbeitern einer Vorgängeraktivität festgelegt werden. So muss ein Arztbrief eben von demjenigen Arzt geschrieben werden, der den Patienten untersucht hat. Er kann nicht von einem beliebigen Benutzer mit der Rolle Arzt erstellt werden. Welche Auswirkungen sich durch solche sogenannte abhängige Bearbeiterzuordnungen für die Skalierbarkeit ergeben, soll in dieser Arbeit untersucht werden. Dabei muss es das Ziel sein, die verteilte WF-Ausführung weiterhin adäquat zu unterstützen, so dass durch die Verwendung von abhängigen Bearbeiterzuordnungen keine besonders hohe Last für die WF-Server und für das Kommunikationssystem entsteht.

1.2.2 Zahlenbeispiel

Das in diesem Abschnitt vorgestellte Zahlenbeispiel belegt, dass die Belastung des Kommunikationssystems in unternehmensweiten WfMS tatsächlich ein Problem darstellt. Zu diesem Zweck wird für den in Abb. 1.3 dargestellten Kreditbearbeitungs-WF die Belastung der Teilnetze und Gateways durch eine Berechnung und durch eine Simulation ermittelt (siehe auch [BD00a]). Weitere Zahlenbeispiele, die zu ähnlichen Ergebnissen führen, finden sich in [BD97, BD98].

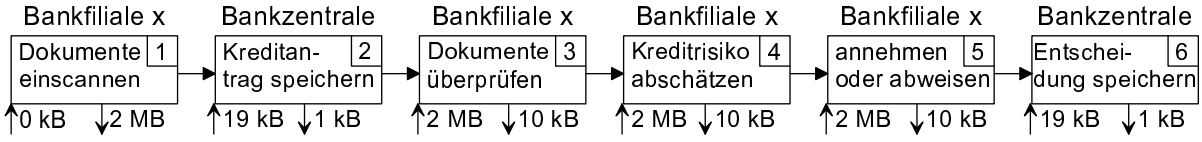


Abbildung 1.3 Beispiel-WF einer Kreditantragsbearbeitung.

In dem in Abb. 1.3 dargestellten Szenario[2] einer Kreditbearbeitung in einer Bank enthalten die Ein- bzw. Ausgabeparameter der Aktivitäten 1, 3, 4 und 5 eingescannte Dokumente. Deshalb wird unterstellt, dass sie durchschnittlich einen Parametertransfer von 2 MB notwenig machen.[3] Die Bank bestehe aus 30 Zweigstellen, die mit der Bankzentrale durch ein WAN verbunden sind. In jeder Zweigstelle seien 10 Angestellte beschäftigt, woraus sich insgesamt 300 Angestellte in den Zweigstellen ergeben. Wenn wir außerdem annehmen, dass ein Angestellter einer Filiale durchschnittlich 5 min = 300 sec für die Bearbeitung einer Aktivität benötigt, so bearbeiten die 300 Filialangestellten durchschnittlich eine Aktivität pro Sekunde. Da dies jeweils den Transfer von 2 MB Parameterdaten erfordert, wird das Teilnetz eines zentralen WF-Servers mit 2 MB/sec = 16 Mbit/sec belastet. Damit wäre ein Ethernet-basiertes Local Area Network (LAN) alleine schon durch die Ausführung von Instanzen dieses einen WF-Typs überlastet. Da die betrachteten Aktivitäten 1, 3, 4 und 5 vom WF-Server in der Bankzentrale kontrolliert und in den Filialen bearbeitet werden, muss von den Gateways dieselbe Datenmenge umgesetzt werden.

Eine Alternative zur Verwendung eines zentralen WF-Servers ist die verteilte WF-Steuerung durch die WF-Server der jeweils involvierten Bankfiliale. Um die in diesem Fall auftretende Belastung des Kommunikationssystems mit der des zentralen Falls vergleichen zu können, wurde die Ausführung des in Abb. 1.3 dargestellten WF für den zentralen und den verteilten Fall simuliert (für Details der verwendeten Simulationsumgebung und der konkreten Simulation siehe Kapitel 9 und Anhang C). Im zentralen Fall ergibt sich bei der Simulation eine Belastung[4] des Teilnetzes des WF-Servers von

[2] Eine detaillierte Beschreibung des Szenarios findet sich im Anhang C.1.

[3] Bei den in dieser Arbeit betrachteten unternehmensweiten Anwendungsbereichen sind durchaus auch wesentlich größere Datenmengen denkbar, z.B. für multimediale Daten.

[4] Diese Last ist geringer als der rechnerische Wert von 16 Mbit/sec, da es bei der Simulation nicht möglich ist, die Bearbeiter voll auszulasten. Dies würde zum „Überlaufen“ der Arbeitslisten dieser Benutzer führen. Aus diesem Grund bearbeiten die Angestellten der Zweigstellen weniger als 1 Aktivität/sec.

15,7 Mbit/sec. Wenn die WF-Steuerung jedoch verteilt durchgeführt wird, so wird kein Teilnetz mit mehr als 0,5 Mbit/sec belastet. Der Engpass im Kommunikationssystem kann auf diese Weise also vermieden werden.

Da (vor allem in weiträumig verteilten Systemen) die Belastung der Gateways ebenfalls sehr relevant ist, wurde auch diese für das betrachtete Szenario simuliert. Dabei ergibt sich für den zentralen Fall, dass die Gateways 15,6 Mbit/sec umzusetzen haben. Findet diese Kommunikation z.B. über ISDN statt, so werden 243,6 ständig benutzte ISDN-Verbindungen (mit je 64 kbit/sec) benötigt. Werden die WF-Instanzen dagegen verteilt gesteuert, so müssen von den Gateways insgesamt nur 80,6 kbit/sec übertragen. Dies entspricht lediglich 1,3 ISDN-Verbindungen.

Mit diesem Zahlenbeispiel konnte gezeigt werden, dass in einem unternehmensweiten WfMS sowohl die Belastung der Teilnetze, wie auch der Gateways, ein Problem darstellen kann. Außerdem wurde an einem Beispiel erläutert, wie sich diese Belastung durch eine verteilte WF-Steuerung drastisch reduzieren lässt.

1.2.3 Rahmenbedingungen

Das Ziel des in dieser Arbeit vorgestellten Ansatzes ist es, den Einsatz von WfMS für Anwendungen zu ermöglichen, bei denen eine sehr große Last entsteht (vgl. Abschnitt 1.2.1). Zu diesem Zweck wird ein Konzept für ein skalierbares WfMS entwickelt, welches z.B. für die Realisierung unternehmensweiter prozessorientierter Anwendungssysteme verwendet werden kann. Die vorgestellten Verfahren können auch für unternehmensübergreifendes WF-Management eingesetzt werden. Allerdings ergeben sich dadurch zusätzliche Anforderungen, die in dieser Arbeit nur am Rande angesprochen werden. So kann erforderlich sein, dass bestimmte vom WfMS verwaltete Daten ein Unternehmen nicht verlassen, oder dass ein Unternehmen sein WfMS autonom konfigurieren und WF-Vorlagen autonom verändern kann.

Der in dieser Arbeit entwickelte Ansatz setzt einige Annahmen voraus, die aber i.d.R. bei prozessorientierten Anwendungssystemen bzw. bei dem zu deren Realisierung verwendeten WfMS erfüllt sind. Dies sind im Einzelnen:

- Das WfMS verfügt über ein Organisationsmodell, in dem Personen sowohl mit OE in der Aufbauorganisation als auch mit Rollen assoziiert werden können.
- Die Zuordnung von Bearbeitern zu einer zur Bearbeitung anstehenden Aktivität erfolgt zur Ausführungszeit mittels eines Auswahlprädikats, wie z.B. `Rolle = Arzt` $\wedge$ `OE = Radiologie`. In diesen Bearbeiterzuordnungen können auch Vorgängeraktivitäten referenziert werden (abhängige Bearbeiterzuordnungen). In beiden Fällen wird angenommen, dass sich der tatsächliche Bearbeiter einer Aktivität zufällig aus den durch die Bearbeiterzuordnung festgelegten Bearbeitern der Aktivität ergibt. Diese Bearbeiter befinden sich nicht notwendigerweise alle im selben Teilnetz.
- Das WfMS verwendet mehrere WF-Server sowie ein Kommunikationssystem, das aus mehreren Teilnetzen besteht. Zur Vereinfachung wird angenommen, dass jedes betrachtete Teilnetz über einen (und nur einen) eigenen WF-Server verfügt, und dass jeder dieser WF-Server im Prinzip für jeden WF-Typen und jeden Aktivitätentypen zur Steuerung eingesetzt werden kann. Der Fall, dass sich mehrere WF-Server im selben Teilnetz befinden, wird in Kapitel 7 gesondert betrachtet. Einem Benutzer des WfMS sind i.d.R. ein oder mehrere Teilnetze fest zugeordnet.

- Die Anzahl der Personen, die eine bestimmte Aktivität bearbeiten dürfen, ihre Zuordnung zu den Teilnetzen und die Topologie des Kommunikationsnetzwerks ändern sich zwischen der Modellierungszeit und der WF-Ausführung nicht signifikant.

1.3 Ziel und Aufbau der Arbeit

Der Ausgangspunkt dieser Arbeit war die Fragestellung, ob es möglich ist, ein WfMS so zu realisieren, dass trotz einer sehr großen Gesamtlast keine Systemkomponente überlastet ist. Außerdem sollen für die notwendige Systeminfrastruktur (Rechner und Kommunikationssystem) keine übermäßig hohen Kosten entstehen. Diese Gesamtfragestellung gliedert sich in eine Reihe von Teilproblemen, die in dieser Arbeit gelöst werden. Die wichtigsten dieser Teilaspekte sind:

- Wie soll das dem WfMS zugrunde liegende System (die Hardware) aufgebaut sein? Gibt es Systemkonfigurationen, mit denen sich das WF-Management besonders effizient realisieren lässt?
- Wie sind die zu bewältigen Aufgaben zu verteilen? Welche Strategie muss verfolgt werden, damit bei der WF-Ausführung eine möglichst geringe Last entsteht?
- Ausgehend von der These, dass es möglich ist, eine günstige Verteilung der Aufgaben auf die WF-Server automatisch zu ermitteln (sofern genügend Information zur Verfügung steht), stellen sich die folgenden Fragen: Wie kann eine gegebene Aufgabenverteilung bewertet werden? Hierzu ist es notwendig, ein Kostenmodell zu entwickeln. Mit welchen Verfahren kann eine geeignete Aufgabenverteilung ermittelt werden? Das naive Durchprobieren aller Möglichkeiten, um die optimale Lösung zu ermitteln, verbietet sich vermutlich, weil der Suchraum zu groß ist.
- Wie können die – wegen der Verteilung der Aufgaben auf mehrere WF-Server notwendig werdenden – Migrationen überhaupt realisiert werden? Wie kann das dabei existierende Optimierungspotential genutzt werden?
- Sind die entwickelten Verfahren auch im Zusammenspiel mit fortschrittlichen WF-Konzepten wie abhängigen Bearbeiterzuordnungen oder dynamischen Änderungen realisierbar? Hier kann ein Problem bzgl. der zur Verfügung stehenden Information entstehen: Einerseits ist bei verteilter WF-Steuerung nicht jedem WF-Server die gesamte Information bekannt, was zu Problemen bei der Realisierung anderer WF-Konzepte führen kann. Andererseits wird zur Berechnung einer geeigneten Aufgabenverteilung Information benötigt, die durch andere Konzepte evtl. verfälscht wird und deshalb nicht mehr beschafft werden kann.
- Wie kann (qualitativ und quantitativ) gezeigt werden, dass die in dieser Arbeit entwickelten Verfahren tatsächlich einen Vorteil gegenüber schon bekannten Verfahren darstellen? Nur wenn dieser Beweis erbracht werden kann, ist das Ergebnis dieser Arbeit tatsächlich für den praktischen Einsatz relevant.

Die aufgezählten Aspekte bauen teilweise aufeinander auf. Der Aufbau dieser Arbeit entspricht deshalb ungefähr ihrer Reihenfolge. Als Abschluss des Motivations- und Einleitungsteils werden im nächsten Kapitel die Grundlagen für das Verständnis der Arbeit geschaffen. So wird das WF-Metamodell von ADEPT vorgestellt, also erläutert, wie der Kontroll- und Datenfluss einer WF-Vorlage spezifiziert wird. Außerdem wird dieses im Prinzip bekannte Modell an einigen Stellen erweitert. Schließlich wird im Kapitel 2 noch beschrieben, wie die Bearbeiter den Aktivitäten zugeordnet werden und auf welche Art und Weise die Arbeitslisten der Benutzer verwaltet werden.

Der Teil II bildet den originären, theoretischen Teil dieser Arbeit. Er klärt, wie effizientes verteiltes WF-Management realisiert werden kann. Dieser Teil beginnt mit Kapitel 3, in welchem die System-

infrastruktur entwickelt wird, die dem WfMS zugrunde liegen soll. Das Ergebnis wird sein, dass diese aus mehreren WF-Servern besteht, die auf verschiedene Teilnetze des Kommunikationssystems verteilt werden. Ausgehend von dieser Struktur ergibt sich eine Strategie für die Aufgabenverteilung, bei der eine WF-Instanz abschnittsweise von mehreren WF-Servern kontrolliert wird. Dies bedingt die Notwendigkeit von Migrationen, deren Grundprinzip ebenfalls in Kapitel 3 geklärt wird. So wird untersucht, zu welchen Zeitpunkten Kommunikationen zwischen den WF-Servern notwendig werden und welche Informationen bei Migrationen übertragen werden müssen.

Die Serverzuordnung einer Aktivität legt fest, von welchem WF-Server diese Aktivität kontrolliert wird. Nun war eine Forderung, dass eine geeignete Verteilung der Aufgaben auf die WF-Server automatisch berechnet werden soll. Deshalb beschäftigt sich das Kapitel 4 mit Verfahren zur Berechnung von geeigneten Serverzuordnungen. Um die optimalen Serverzuordnungen für die Aktivitäten einer WF-Vorlage zu ermitteln, wäre ein exponentieller Aufwand notwendig. Da WF-Vorlagen sehr viele Aktivitäten enthalten können, ist dies nicht akzeptabel. Deshalb werden im Kapitel 4 Verfahren zur Berechnung von gut geeigneten Serverzuordnungen vorgestellt. Damit es überhaupt möglich ist, geeignete Serverzuordnungen automatisch zu berechnen, wird ein Kostenmodell für die Bewertung der Serverzuordnungen einer WF-Vorlage benötigt. Dieses wird realisiert, indem ausgehend von der Lokation des Servers und der potentiellen Bearbeiter einer Aktivität gewisse Wahrscheinlichkeitsverteilungen berechnet werden. Mit diesen lassen sich dann die resultierenden Kosten abschätzen.

Die in Kapitel 4 entwickelten Verfahren sollen auch im Zusammenspiel mit abhängigen Bearbeiterzuordnungen (z.B. `selber Bearbeiter wie Aktivität m`) eingesetzt werden können. Dann stehen die Bearbeiter, die sich für die Ausführung einer bestimmten Aktivität qualifizieren, zur Modellierungszeit jedoch noch nicht fest. In diesem Fall sind zur Modellierungszeit vorberechnete statische Serverzuordnungen i.d.R. ungeeignet. Deshalb werden im Kapitel 5 variable Serverzuordnungen eingeführt und es wird geklärt, wie eine verteilte WF-Ausführung mit diesen realisiert werden kann. Außerdem wird aufgezeigt, wie es auch bei WF-Vorlagen mit abhängigen Bearbeiterzuordnungen und variablen Serverzuordnungen möglich ist, die zur Abschätzung der entstehenden Kosten notwendigen Wahrscheinlichkeitsverteilungen zu berechnen. Dadurch lassen sich auch geeignete variable Serverzuordnungen automatisch ermitteln.

Im Kapitel 6 wenden wir uns nochmals den Migrationen zu. Deren Ausführung verursacht hohe Kommunikationskosten, da mit dem internen Zustand und den Anwendungsdaten der WF-Instanz eine sehr große Datenmenge übertragen werden muss. Deshalb wird im Kapitel 6 untersucht, wie die Durchführung von Migrationen optimiert werden kann. Dazu wird das zu übertragende Datenvolumen durch zur Ausführungszeit durchgeführte Maßnahmen reduziert. Diese verhindern die redundante Übertragung sowohl der Zustands- als auch der Anwendungsdaten der WF-Instanz.

In den Kapiteln 3 bis 6 wird der Aspekt der Reduzierung der in einem WfMS entstehenden Kommunikationslast betrachtet. Durch die entsprechenden Verfahren soll eine Überlastung des Kommunikationssystems vermieden werden. Ein WF-Server kann jedoch auch dann überlastet sein, wenn das zugehörige Teilnetz nicht überlastet ist. Deshalb werden im Kapitel 7 Verfahren entwickelt, die erlauben, dass sich mehrere WF-Server in demselben Teilnetz befinden. Diese Server können sich dann die Last teilen. Die Besonderheit des vorgestellten Ansatzes ist, dass er bei der WF-Ausführung keine zusätzliche Kommunikation erfordert. Deshalb werden die in den vorangehenden Kapiteln erzielten Errungenschaften, die zu einer Reduzierung der Kommunikationslast führen, durch dieses Verfahren nicht beeinträchtigt.

Den Abschluss des Teils II bildet das Kapitel 8. In diesem wird untersucht, wie die verteilte WF-Ausführung mit dem Konzept dynamischer Änderungen von WF-Instanzen zu deren Ausführungszeit

vereinbar ist. Um die Konsistenz der aus einer Änderung resultierenden WF-Instanz garantieren zu können, erfordern dynamische Änderungen eine zentrale Kontrollinstanz. Die Existenz einer solchen widerspricht aber diametral den Anforderungen an verteilte WF-Ausführung, falls diese auf eine effiziente Art und Weise erfolgen soll. Im Kapitel 8 wird gezeigt, wie die für verteilte dynamische Änderungen notwendigen Synchronisationen so realisiert werden können, dass für das Kommunikationssystem wenig zusätzliche Last entsteht. Außerdem betrachten wir, wie eine effiziente verteilte Ausführung von solchen WF-Instanzen möglich ist, die zuvor verändert wurden.

Der Teil III beinhaltet die Zusammenfassung der Arbeit und ordnet diese in die bestehende Literatur ein. Im Kapitel 9 wird die zum Thema verteilte WfMS erschienene Literatur diskutiert. Dazu werden die entsprechenden Ansätze klassifiziert. Die vorliegende Arbeit wird in die Klassifikation eingeordnet und mit den anderen Ansätzen kritisch verglichen. Um zu zeigen, wie groß die Vorteile von ADEPT$_{distribution}$ tatsächlich sind, ist ein quantitativer Vergleich erforderlich. Deshalb wird das entwickelte Modell mit anderen Typen verteilter WfMS verglichen. Dies erfolgt, indem durch Simulationen für mehrere Szenarien die bei den verschiedenen Verteilungsmodellen entstehende Last ermittelt wird. Dabei zeigt sich, dass ADEPT$_{distribution}$ bei bestimmten Szenarien eine beträchtliche Reduktion der Last ermöglicht.

Die eigentliche Diskussion der Arbeit findet in Kapitel 10 statt. Dort wird aufgezeigt, welches die originären Beiträge dieser Arbeit zur WF-Forschung sind. Hierzu werden die im Teil II bearbeiteten Themen nochmals betrachtet, wobei die Neuerungen jeweils im Kontext existierender Literatur diskutiert werden.

Im Kapitel 11 wird aufgezeigt, wie weit die Implementierung der in dieser Arbeit vorgestellten Konzepte fortgeschritten ist. Das Kapitel 12 beendet diese Arbeit mit einem Resümee der vorgefundenen Probleme und der in dieser Arbeit erzielten Ergebnisse und fasst die Hauptergebnisse der Arbeit nochmals kurz zusammen.

Der Anhang A enthält eine Erläuterung der in dieser Arbeit eingeführten Abkürzungen, Funktionen und Ausdrücke. Er soll es ermöglich, deren Bedeutung schnell nachzuschlagen. Im Anhang B finden sich einige Beweise, die aus der Arbeit ausgelagert wurden, um die Lesbarkeit zu erhöhen. Der Anhang C bietet eine detaillierte Beschreibung des in Abschnitt 1.2.2 eingeführten Zahlenbeispiels.

Alle am Anfang dieses Abschnitts aufgezählten Teilprobleme werden im Rahmen dieser Arbeit gelöst. Damit lässt sich folgern, dass die der Arbeit zugrunde liegende Gesamtfragestellung positiv beantwortet wird. Es ist also möglich, ein skalierbares WfMS effizient zu realisieren.

Kapitel 2

Workflow-Metamodell

In diesem Kapitel werden Klassen von WfMS gebildet, indem der Informationsgehalt der verschiedenen denkbaren WF-Metamodelle (im Folgenden kurz WF-Modelle genannt) untersucht wird. Dann wird analysiert, zu welcher dieser Klassen ein für diese Arbeit geeignetes WF-Modell gehören sollte. Anschließend wird ein entsprechendes WF-Modell vorgestellt. Es dient als Grundlage, um in den nachfolgenden Kapiteln Algorithmen formal beschreiben zu können. Dieses WF-Modell ADEPT$_{base}$ stellt keine Einschränkung für die Allgemeingültigkeit der in dieser Arbeit gemachten Aussagen dar, sondern es ist als Diskussionsgrundlage zu verstehen. Fast alle vorgestellten Verfahren können ebenso auf anderen (z.B. Petri-Netz-basierten) WF-Modellen (wie FunSoft-Netzen [DG94]) angewendet werden. ADEPT ist ein sehr fortschrittlicher Ansatz für WF-Management, bei dem z.B. die dynamische Veränderung der Struktur von WF-Instanzen zu deren Ausführungszeit möglich ist (siehe Kapitel 8). Falls es gelingt, effizientes WF-Management für ADEPT zu realisieren, so ist dies auch bei einfacheren WF-Modellen möglich, da diese Einschränkungen mit sich bringen (wie z.B. das Fehlen von Schleifen im Modell von IBM FlowMark[1] [IBM96b, LA94, LR94, LR00]), welche die zu lösende Aufgabe eher erleichtern.

Das ADEPT-Basismodell ADEPT$_{base}$ wird in [Rei00] formal definiert und Details können dort bei tiefergehendem Interesse nachgelesen werden. Im Folgenden wird es an einigen Stellen vereinfacht und nur in soweit vorgestellt, wie es für das Verständnis dieser Arbeit notwendig ist. Um Platz zu sparen, ist die Darstellung nicht so formal wie in [Rei00]. Das WF-Modell wurde für die vorliegende Arbeit an einigen Stellen erweitert. Auch diese Erweiterungen werden in dem vorliegenden Kapitel vorgestellt. Außerdem werden solche Aspekte ausführlicher untersucht, die für eine effiziente verteilte WF-Steuerung besonders relevant sind.

In Abschnitt 2.1 werden Klassen von WF-Modellen vorgestellt und eine für diese Arbeit geeignete Klasse ausgewählt. In Abschnitt 2.2 wird beschrieben, wie der Kontrollfluss in ADEPT$_{base}$ festgelegt wird und welche Ausführungssemantik ein WF-Ausführungsgraph hat. Abschnitt 2.3 betrachtet Aspekte des Datenflusses im Allgemeinen und die Realisierung des Datenflusses im ADEPT-Modell im Besonderen. Wie die Bearbeiter den Aktivitäten zugeordnet werden, wird in Abschnitt 2.4 beschrieben. In Abschnitt 2.5 werden mögliche Vorgehensweisen beim Aktualisieren der Arbeitslisten der Benutzer verglichen und es wird ein geeignetes Verfahren für ADEPT entwickelt. Das Kapitel schließt mit einer Zusammenfassung in Abschnitt 2.6.

[1] Der Name der Produkts FlowMark wurde in MQSeries Workflow geändert.

2.1 Vorüberlegungen zum Workflow-Metamodell

Der nun folgende Abschnitt soll dazu beitragen, beurteilen zu können, für welche Arten von WfMS die in dieser Arbeit vorgestellten Verfahren geeignet sind, d.h. welche Anforderungen das entsprechende WF-Metamodell erfüllen muss. Um dies zu ermöglichen, werden zuerst mehrere Klassen von WfMS vorgestellt. In der anschließenden Diskussion wird dann untersucht, bei welchen dieser Klassen die in der vorliegenden Arbeit beschriebenen Verfahren eingesetzt werden können.

2.1.1 Klassen von Workflow-Management-Systemen

Groupware-Systeme, wie z.B. Lotus Notes [DS97], wurden schon in Abschnitt 1.1.3 eingeführt. Dort wurde auch erläutert, wie solche Systeme zur Steuerung von WF-Instanzen eingesetzt werden können. Dabei fällt auf, dass bei Groupware-Systemen den Ausführungskomponenten der Kontrollfluss der WF-Instanzen nicht explizit bekannt ist. Außerdem muss auch keine Information über die potentiellen Bearbeiter der Aktivitäten bekannt sein, da diese von den Benutzern manuell festgelegt oder von Skript-Programmen berechnet werden können.

Bei **dokumentenorientierten WfMS**, wie z.B. ProMInanD [KR91b, Kar94], wird zur Modellierungszeit der Kontrollflussgraph ebenso explizit festgelegt, wie die Bearbeiterzuordnungsausdrücke für die einzelnen Aktivitäten. Allerdings findet i.d.R. keine Definition des Datenflusses statt. Stattdessen ist jeder WF-Instanz eine Menge von Dokumenten zugeordnet, die auch als Elektronische Umlaufmappe bezeichnet wird. Alle diese Dokumente werden jeder einzelnen Aktivität zur Verfügung gestellt, und jede Aktivität hat die Möglichkeit, beliebige Dokumente zu verändern. Durch diese Vorgehensweise entfällt die Notwendigkeit für die Spezifikation der Ein- und Ausgabedaten der Aktivitätenprogramme.

Produktions-WfMS [LR00] sind für die Steuerung der für ein Unternehmen besonders wichtigen Anwendungen geeignet. Zu diesen gehören sicherlich auch unternehmensweite und -übergreifende Anwendungssysteme. Um einen hohen Grad an Kontrolle ausüben zu können, benötigen solche Systeme möglichst viel Information. Deshalb werden in einer WF-Vorlage nicht nur der Kontrollfluss und die Bearbeiterzuordnungen spezifiziert, sondern es muss auch der gesamte Datenfluss (inkl. Datentypen) festgelegt werden.

Die Zugehörigkeit eines WfMS zu dieser Klasse erzwingt keine bestimmte Art der Kontrollflussmodellierung. So gehören auf Petri-Netzen [Obe96] basierende Ansätze wie FunSoft-Netze [DG94] ebenso zu dieser Klasse, wie das auf gefärbten Graphen basierende FlowMark [LA94]. Auch eine Kontrollflussspezifikation mittels State- und Activitycharts, wie sie z.B. von MENTOR [WWWK96b, MWW+98] verwendet wird, erfüllt die an diese Klasse gestellten Anforderungen.

2.1.2 Einordnung von ADEPT$_{distribution}$

Im Folgenden wird untersucht, welche der soeben beschriebenen Klassen von WfMS geeignet sind, um ein effizientes skalierbares WF-Management zu realisieren. Dazu analysieren wir, ob die jeweils zur Verfügung stehende Information ausreichend ist, um dieses Ziel zu erreichen. Anschließend wird diskutiert, wie das in dieser Arbeit verwendete WF-Metamodell gestaltet werden muss, um einerseits genügend Information zur Lösung der Problemstellung zur Verfügung zu stellen, andererseits die Verwendbarkeit der zu entwickelnden Verfahren nicht dadurch zu beschränken, dass Information vorausgesetzt wird, die in WfMS der ausgewählten Klasse(n) normalerweise nicht verfügbar ist.

Wie schon in Abschnitt 1.1.3 erläutert wurde, sind Groupware-basierte WfMS i.d.R. nicht geeignet, um große prozessorientierte Anwendungssysteme zu realisieren. Der Grund dafür ist, dass zur Ausführungszeit der WF-Instanzen nicht genügend Information zur Verfügung steht, um eine zuverlässige WF-Ausführung zu gewährleisten. Dies gilt insbesondere, wenn fortschrittliche Konzepte, wie dynamische Ablaufänderungen oder die Überwachung von Zeitbedingungen unterstützt werden sollen. Deshalb ist es auf Basis solcher Systeme nicht möglich, ein unternehmensweites WfMS zu realisieren, weshalb diese Klasse nicht weiter berücksichtigt wird.

Dokumentenorientierte WfMS haben den Schwachpunkt, dass keinerlei Information über den Datenfluss verfügbar ist. Da deshalb stets alle Dokumente zu jedem Aktivitätenprogramm übertragen werden müssen, ist schon die zentrale WF-Steuerung äußerst ineffizient. WfMS dieser Klasse sind erst recht nicht geeignet, um ein effizientes skalierbares WfMS zu realisieren. Allerdings sind mit dem Kontrollflussgraphen und den Bearbeiterzuordnungen in dokumentenorientierten WfMS wichtige Informationen bekannt, so dass einige der in dieser Arbeit vorgestellten Verfahren auch auf WfMS dieser Klasse übertragbar sind. Dokumentenorientierte WfMS liegen aber nicht im Fokus dieser Arbeit.

Produktions-WfMS verfügen mit dem Kontrollfluss, dem Datenfluss und den Bearbeiterzuordnungen der Aktivitäten über diejenigen Informationen, die von den meisten der in dieser Arbeit vorgestellten Verfahren benötigt werden. Da es sich bei Produktions-WfMS außerdem um diejenigen Systeme handelt, die für den Einsatz in großen Anwendungssystemen vorgesehen sind, wird im Folgenden davon ausgegangen, dass ein WF-Modell die soeben erwähnten Informationen beinhaltet. Diese Informationen spiegeln sich also auch in dem in den nachfolgenden Abschnitten vorgestellten ADEPT-Metamodell wider.

Außer den schon erwähnten Informationen verwenden die in dieser Arbeit vorgestellten Verfahren auch noch eine Historie des Ablaufs der betrachteten WF-Instanz. Diese Historie entspricht in etwa einer Log-Datei, in der für die WF-Instanz relevante Ereignisse protokolliert werden. Ein solches „Log“ ist in vielen Produktions-WfMS ohnehin schon (aus Gründen der Nachvollziehbarkeit) verfügbar. Andernfalls kann es zusätzlich erzeugt werden, ohne dass dies (negative) Auswirkungen auf andere WF-Konzepte hat. Deshalb wird im ADEPT-Metamodell von der Existenz einer solchen Ablaufhistorie ausgegangen.

Wie wir noch sehen werden, bietet das ADEPT-Modell auch noch eine Art von Information, die in Produktions-WfMS üblicherweise nicht verfügbar ist. Dabei handelt es sich um Blöcke von Kontrollflusskonstrukten, die in Abschnitt 2.2.1 eingeführt werden. Die Existenz dieser Blöcke ermöglicht es, bestimmte Verfahren einfacher und effizienter zu gestalten. Die entsprechende Information wird in der vorliegenden Arbeit aber nur an extrem wenigen Stellen verwendet, an denen auch gesondert auf diese Tatsache hingewiesen wird. Außerdem wird dann erläutert, wie sich das entsprechende Problem unter Zugrundelegung anderer WF-Modelle lösen lässt.

Zusammenfassend sei noch einmal klargestellt, dass für $\text{ADEPT}_{distribution}$ ein WF-Metamodell gewählt wird, das prinzipiell auf alle Produktions-WfMS übertragbar ist. Dass WfMS der anderen Klassen nur bedingt (dokumentenorientierte WfMS) oder überhaupt nicht (Groupware-basierte WfMS) unterstützt werden, stellt keine echte Einschränkung für die Allgemeingültigkeit der in dieser Arbeit gemachten Aussagen dar, da solche Systeme für die Realisierung von skalierbaren WF-Anwendungen ohnehin nicht geeignet sind. In den nun folgenden Abschnitten wird das im Weiteren verwendete WF-Modell ADEPT_{base} vorgestellt, das die oben beschriebenen Anforderungen erfüllt.

2.2 Kontrollfluss

In diesem Abschnitt wird erläutert, wie der Kontrollfluss in ADEPT$_{base}$ realisiert wird. Zu diesem Zweck werden die Konstrukte vorgestellt, mit denen der Kontrollfluss modelliert wird und es wird ihr Verhalten bei der WF-Ausführung definiert. Dabei wird bei ADEPT$_{base}$ das Ziel verfolgt, dass möglichst viel Information über den Zustand einer WF-Instanz verfügbar ist (vgl. [RD98, Rei00]). So kann im Gegensatz zu Petri-Netz-basierten Ansätzen meist direkt am Zustand der WF-Instanz erkannt werden, ob eine bestimmte Aktivität schon ausgeführt wurde oder nicht.

2.2.1 Workflow-Vorlagen

Der Kontrollfluss spezifiziert die Ausführungsreihenfolge der Aktivitäten und damit der ihnen zugeordneten Aktivitätenprogramme. In ADEPT$_{base}$ wird bei der Kontrollflussdefinition ein blockorientierter Ansatz verfolgt. Wie in Abb. 2.1 dargestellt, bildet jedes Kontrollflusskonstrukt einen Block mit je genau einer eindeutigen Start- und Endeaktivität. Diese Blöcke können verschachtelt sein. Auch ein kompletter WF-Typ bildet einen Block mit einer eindeutigen Start- und Endeaktivität ($Start$ und End in Abb. 2.1). Dies sind sogenannte Null-Aktivitäten, denen keine Aktivitätenvorlagen zugeordnet sind, so dass beim Starten dieser Aktivitäten auch kein Aktivitätenprogramm ausgeführt wird. Die blockorientierte Modellierung wurde gewählt, da sie für die Benutzer leicht verständlich ist und außerdem eine effiziente Überprüfung bestimmter Modelleigenschaften, wie z.B. Zyklenfreiheit, ermöglicht. In den nun folgenden Unterabschnitten wird die Bedeutung der in Abb. 2.2 dargestellten Kontrollflusskonstrukte von ADEPT$_{base}$ erläutert.

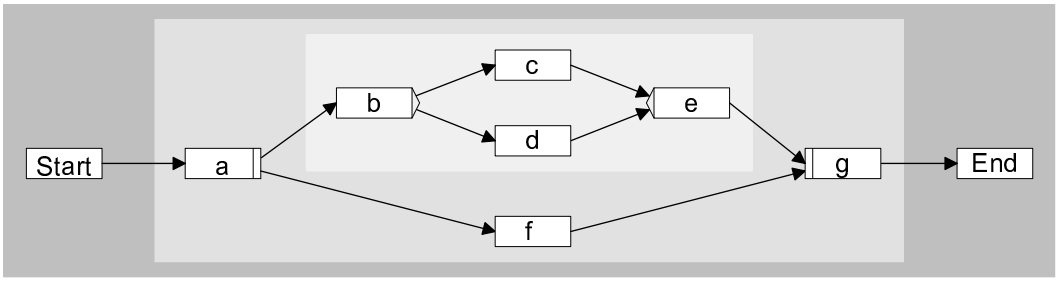


Abbildung 2.1 Blockstrukturierung in ADEPT$_{base}$.

2.2.1.1 Sequenz

Das einfachste Kontrollflusskonstrukt von ADEPT$_{base}$ ist die Sequenz. Eine solche ist in Abb. 2.2a dargestellt und legt fest, dass die Aktivität b nach der Aktivität a ausgeführt wird. Die beiden Aktivitäten werden durch eine Kante vom Typ (engl. edge type) $ET = CONTROL\_E$ verbunden. Wenn in dieser Arbeit von Kontrollkanten gesprochen wird, sind stets Kanten vom Typ $ET = CONTROL\_E$ gemeint und die in den Abbildungen verwendeten Kanten sind von diesem Typ, sofern nichts anderes angegeben ist. Die in eine Sequenz involvierten Aktivitäten haben die Ausgangssemantik $V_{out}(a) =$ *ONE_Of_ONE* und die Eingangssemantik $V_{in}(b) =$ *ONE_Of_ONE*. Da die Aktivitäten a und b in Abb. 2.2a zugleich die Start- und Endeaktivitäten des WF repräsentieren (mit Knotentyp (engl. node type): $NT(a) =$ *STARTFLOW*, $NT(b) =$ *ENDFLOW*), haben sie die Eingangssemantik $V_{in}(a) =$ *NONE* bzw. $V_{out}(b) =$ *NONE* („normale“ Aktivitäten sind vom Knotentyp $NT =$ *ACTIVITY*).

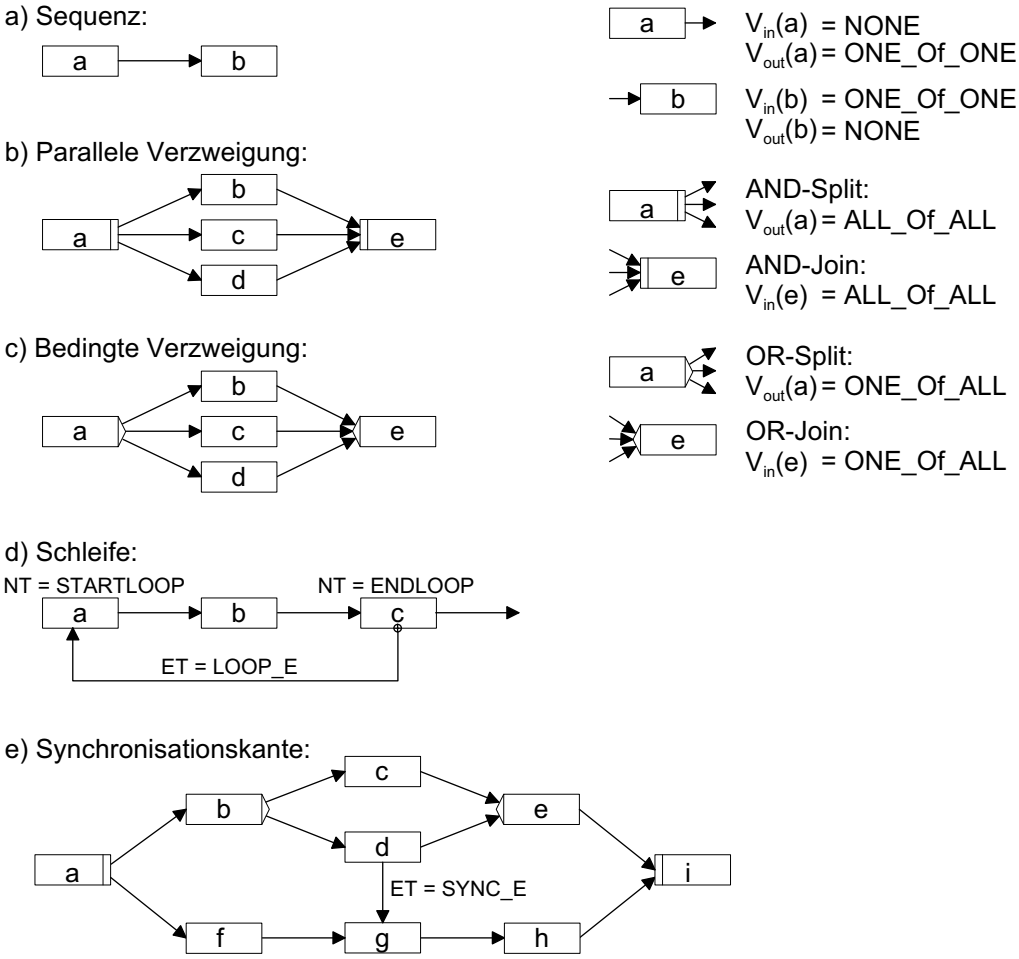


Abbildung 2.2 Kontrollflusskonstrukte von ADEPT$_{base}$.

2.2.1.2 Verzweigungen

Bei Verzweigungen (engl. split) unterscheiden wir parallele und bedingte Verzweigungen. Bei der in Abb. 2.2b dargestellten parallelen Verzweigung werden nach Beendigung der AND-Split-Aktivität a, wegen der Ausgangssemantik $V_{out}(a) = ALL\_Of\_ALL$ alle drei parallelen Zweige ausgeführt. Die AND-Join-Aktivität e kann erst dann ausgeführt werden, wenn alle parallelen Zweige beendet wurden. Dahingegen ist Abb. 2.2c ein Beispiel für eine bedingte Verzweigung. Mit der Aktivität a ist ein Verzweigungsprädikat verbunden, das nach Beendigung dieser Aktivität ausgewertet wird. Von dessen Ergebnis ist abhängig, welcher Zweig zur Ausführung ausgewählt wird.

2.2.1.3 Schleifen

In Abb. 2.2d ist eine Schleife dargestellt. Diese hat eindeutige Start- und Endeknoten (die Aktivitäten a und c), mit den Knotentypen $NT(a) =$ *STARTLOOP* und $NT(c) =$ *ENDLOOP*. Diese sind durch eine Kante e vom Typ $ET(e) = LOOP\_E$ verbunden. Mit dem Schleifen-Endeknoten ist eine Bedingung verknüpft, die angibt, ob die Schleife ein weiteres Mal durchlaufen werden oder über die

normale Kontrollkante verlassen werden soll. Das Verhalten einer Schleife in ADEPT$_{base}$ ist also ähnlich zu dem einer Repeat-Until-Schleife in einer Programmiersprache.

2.2.1.4 Synchronisationskanten

Die durch die bisher vorgestellten Konstrukte erzwungene Blockstrukturierung stellt eine Einschränkung bei der Modellierung dar. Um die in der Praxis benötigte Ausdrucksmächtigkeit des WF-Modells zu erhalten, werden deshalb Synchronisationskanten eingeführt. In dem Beispiel aus Abb. 2.2e erzwingt die Synchronisationskante von Aktivität d zu g, dass die Aktivität g nicht vor d ausgeführt werden kann. Das heißt, die Aktivität g kann erst ausgeführt werden, wenn die Aktivität d beendet ist, oder wenn sichergestellt ist, dass d nicht ausgeführt wird. Letzteres ist in diesem Beispiel dann der Fall, wenn bei der an Aktivität b beginnenden bedingten Verzweigung der obere Zweig gewählt wurde.

2.2.1.5 ADEPT-Kontrollflussgraph

Mit den in Abschnitt 2.2.1.1 bis 2.2.1.4 vorgestellten Konstrukten wird in ADEPT$_{base}$ der Kontrollfluss eines WF-Typs definiert. Formal lässt sich ein WF damit wie in Definition 2.1 angegeben spezifizieren. In der vorliegenden Arbeit nicht betrachtete Aspekte eines Kontrollflussgraphen sind in Definition 2.1 nicht berücksichtigt und können bei Interesse in [Rei00] nachgelesen werden.

Definition 2.1 (ADEPT-Kontrollflussgraph)
Ein ADEPT-Kontrollflussgraph ist ein Tupel $CFS = (N, E, D, NT, V_{out}, V_{in}, \ldots)$ mit
Knotenmenge N (den Aktivitäten),
Kantenmenge $E \subseteq N \times N \times EdgeTypes$ mit
$EdgeTypes := \{CONTROL\_E, LOOP\_E, SYNC\_E\}$,
Datenelementen D,
Knotentypen $NT : N \mapsto NodeTypes$ mit
$NodeTypes :=$ {*ACTIVITY, NULL*[2], *STARTFLOW, ENDFLOW, STARTLOOP, ENDLOOP*},
Ausgangssemantik der Knoten $V_{out} : N \mapsto OutBehaviour$ mit
$OutBehaviour :=$ {*NONE, ONE_Of_ONE, ALL_Of_ALL, ONE_Of_ALL*},
Eingangssemantik der Knoten $V_{in} : N \mapsto InBehaviour$ mit
$InBehaviour :=$ {*NONE, ONE_Of_ONE, ALL_Of_ALL, ONE_Of_ALL*}

2.2.2 Ausführung von Workflow-Instanzen

Um Instanzen eines WF-Typs ausführen zu können, muss deren Verhalten zur Ausführungszeit formal definiert sein. Zu diesem Zweck werden den Aktivitäten und Kanten des Kontrollflussgraphen einer WF-Instanz Zustände (Markierungen) zugeordnet, die sich im Laufe der Lebenszeit der WF-Instanz verändern. Im Folgenden wird erläutert, welche Zustände dies sind und nach welchen Regeln sie sich verändern. Diese Regeln definieren die Ausführungssemantik einer WF-Instanz in dem WF-Modell ADEPT$_{base}$.

[2] $NT = NULL$ steht für Null-Aktivitäten mit denen (ebenso wie mit den Start- ($NT = STARTFLOW$) und Endeknoten ($NT = ENDFLOW$) des WF), kein Aktivitätenprogramm verbunden ist.

2.2.2.1 Zustände von Aktivitäten und Kanten

Der Statechart aus Abb. 2.3 zeigt die Zustände und Zustandsübergänge, die während der Ausführung einer elementaren Aktivitäteninstanz auftreten können. Initial befindet sich jede Aktivität im Zustand NOT_ACTIVATED, d.h. die Aktivität ist noch nicht bearbeitbar. Sind alle Vorbedingungen für die Bearbeitung der Aktivität erfüllt (s.u.), so wechselt der Zustand in ACTIVATED. Handelt es sich um eine automatische Aktivität (d.h. die Aktivität wird vom System automatisch bearbeitet, ohne dass ein Benutzer involviert ist), so wechselt ihr Zustand direkt in RUNNING. Im Falle einer interaktiven Aktivität wird sie zuerst allen Bearbeitern angeboten, die sich für ihre Ausführung qualifizieren, wobei sie sich im Zustand SELECTED befindet. Erst wenn einer dieser Bearbeiter die Aktivität aus seiner Arbeitsliste ausgewählt hat, geht der Zustand in RUNNING über. Bei Erreichen dieses Zustandes wird das zugehörige Aktivitätenprogramm gestartet. Nach dessen Beendigung geht der Zustand, in Abhängigkeit vom Ergebnis der Aktivitätenbearbeitung, in einen der Zustände COMPLETED oder FAILED über. Diese beiden Zustände werden zum Zustand TERMINATED zusammengefasst. Die Zustände NOT_ACTIVATED, ACTIVATED und SELECTED, bei denen die Aktivitäteninstanz noch nicht gestartet wurde, bilden den Zustand WAITING. Befindet sich eine Aktivität in einem nicht ausgewählten Zweig einer parallelen Aufspaltung, so geht ihr Zustand von WAITING nach SKIPPED über, was anzeigt, dass diese Aktivität nicht ausgeführt wurde.

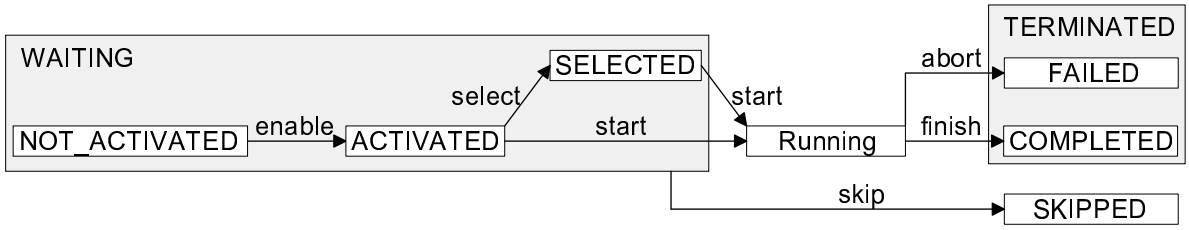


Abbildung 2.3 Statechart einer Aktivitäteninstanz (vereinfacht).

Ob der Zustand einer Aktivitäteninstanz vom Zustand NOT_ACTIVATED nach ACTIVATED wechseln darf, hängt davon ab, ob die Vorbedingungen der Aktivität erfüllt sind. Um dies analysieren zu können, werden Kantenzustände eingeführt. Diese werden von Regeln verwendet, mit denen die sich neu ergebenden Zustände der Knoten berechnet werden (siehe Abschnitt 2.2.2.2). Der Zustand einer Kante ist initial NOT_SIGNALED und geht aufgrund derselben Regeln in TRUE_SIGNALED oder FALSE_SIGNALED über. Abb. 2.4 fasst die Zustände der Knoten und Kanten und deren graphische Darstellung zusammen.

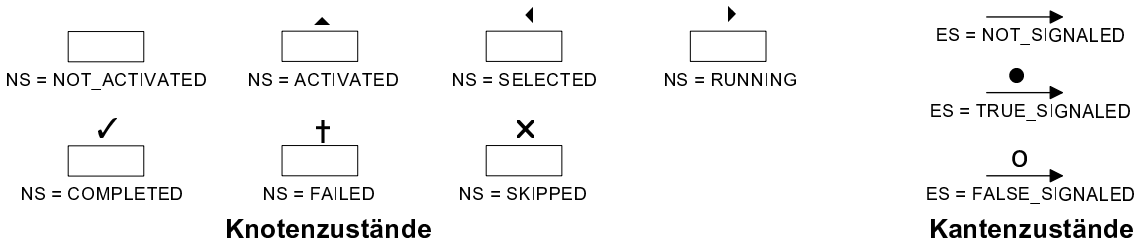


Abbildung 2.4 Graphische Darstellung der Knoten- und Kantenzustände.

Für den Kontrollflussgraphen $CFS = (N, E, \ldots)$ ist die Knotenmarkierung eine Abbildung $NS : N \mapsto NodeStates$ mit $NodeStates$:= {NOT_ACTIVATED, ACTIVATED, SELECTED, RUNNING, COMPLETED, FAILED, SKIPPED}. Diese Abbildung ordnet jedem Aktivitätenknoten $n \in$

N einen Zustand $NS(n) \in NodeStates$ zu. Damit beschreibt die Knotenmarkierung den aktuellen Zustand der Aktivitäten einer WF-Instanz. Der Ausführungsstatus einer Aktivität n wird außerdem durch die Anzahl ihrer bisherigen Durchführungen $It(n) \in \{0, 1, 2, \ldots\}$ ($It : N \mapsto \{0, 1, 2, \ldots\}$) beschrieben. $It(n)$ wird auch als Iterationszähler bezeichnet und bei jeder Beendigung von Aktivität n um eins erhöht. Er wird benötigt, um die aus verschiedenen Schleifeninterationen stammenden Ausführungen von Aktivität n unterscheiden zu können. Damit ergibt sich die Identifikation einer Aktivitäteninstanz $a =< n, it >$ aus dem zugehörigen Aktivitätentyp $n \in N$ und dem Iterationszähler it der Aktivitäteninstanz, wobei $it = It(n)$ der bei der Ausführung dieser Aktivitäteninstanz gültige Wert des Iterationszählers der Aktivität n ist. Schließlich ordnet die Abbildung $ES : E \mapsto EdgeStates$ mit $EdgeStates$:= {NOT_SIGNALED, TRUE_SIGNALED, FALSE_SIGNALED} noch jeder Kante $e \in E$ eine Kantenmarkierung $ES(e) \in EdgeStates$ zu. Damit ergibt sich der Kontrollfluss-Ausführungsgraph einer WF-Instanz durch das 4-Tupel:
$CFS_{instance} = (CFS, NS, ES, It)$

2.2.2.2 Regeln für Zustandübergänge

Die Ausführungssemantik einer WF-Instanz wird durch Ausführungs- und Markierungsregeln formal definiert. Die Ausführungsregeln legen fest, welche Aktivitäten aufgrund der gegebenen Kantenmarkierung ES ausgeführt werden können (also in den Zustand ACTIVATED wechseln dürfen). Dabei gibt die Eingangssemantik der Aktivität n an, ob alle eingehenden Kontrollkanten den Zustand $ES(E)$ = TRUE_SIGNALED haben müssen (bei $V_{in}(n) = ALL\_Of\_ALL$) oder ob es genügt, dass eine Kontrollkante diesen Zustand erreicht hat und die anderen Kanten sich nicht mehr im Zustand NOT_ACTIVATED befinden (bei $V_{in}(n) =$ *ONE_Of_ALL* oder $V_{in}(n) =$ *ONE_Of_ONE*). Die Markierungsregeln legen in Abhängigkeit von der Ausgangssemantik einer Aktivität fest, welche ausgehenden Kanten bei Beendigung der Aktivität mit TRUE_SIGNALED und welche mit FALSE_SIGNALED markiert werden müssen. Diese Regeln werden in [Rei00] formal definiert, weshalb sie im Folgenden nur an einigen Beispielen erläutert werden.

Da im Beispiel aus Abb. 2.5a die Aktivität b beendet wurde ($NS(b)$ = COMPLETED), kann die Kante zur Aktivität c mit TRUE_SIGNALED markiert werden. Dies gilt, weil die Aktivität b die Ausgangssemantik $V_{out}(b) =$ *ONE_Of_ONE* hat. Da die einzige in Aktivität c einmündende Kontrollkante ($V_{in}(c) =$ *ONE_Of_ONE*) nun mit TRUE_SIGNALED markiert ist, darf die Aktivität c nun gestartet werden (Zustandswechsel zu $NS(c)$ = ACTIVATED). Nach Beendigung der Aktivität b kann außerdem die zur Aktivität e des parallelen Zweiges ausgehende Synchronisationskante mit TRUE_SIGNALED markiert werden, da ihre Startaktivität b „normal“ beendet wurde (und nicht ausgelassen wurde). Die Endeaktivität e der Synchronisationskante darf jedoch noch nicht aktiviert werden, da die von Aktivität d einmündende Kontrollkante noch nicht mit TRUE_SIGNALED markiert wurde.

Abbildung 2.5 Beispiele für die Anwendung von Ausführungs- und Markierungsregeln.

Bei dem in Abb. 2.5b dargestellten Beispiel wurde die Startaktivität a einer bedingten Verzweigung soeben beendet. Aufgrund des Verzweigungsprädikats der Aktivität a wird entschieden, welcher

Zweig ausgeführt wird. In unserem Beispiel sei dies der obere Zweig. Deshalb wird die Kante zur Aktivität b mit TRUE_SIGNALED markiert, weshalb die Aktivität b aktiviert werden kann. Die Kanten des unteren Zweiges ($a \rightarrow c$ und $c \rightarrow d$) werden mit FALSE_SIGNALED und die Aktivitäten dieses Zweiges (Aktivität c) werden mit SKIPPED markiert. Dieses Vorgehen entspricht der Dead-Path-Elimination, wie sie z.B. bei IBM FlowMark verwendet wird [LA94, IBM96b]. Gehen von einem nicht gewählten Zweig einer bedingten Aufspaltung Synchronisationskanten aus, so werden diese mit FALSE_SIGNALED markiert, was aber nicht verhindert, dass die Zielaktivität (später) aktiviert werden kann (falls die eingehende Kontrollkante mit TRUE_SIGNALED markiert wird). Synchronisationskanten sollen, wie in Abschnitt 2.2.1.4 festgelegt wurde, sicherstellen, dass ihre Zielaktivität erst ausgeführt werden kann, wenn ihre Startaktivität beendet wurde oder nicht mehr ausgeführt werden kann. Um das Starten der Zielaktivität einer Synchronisationskante im letzteren Fall zu ermöglichen, darf sie auch aktiviert werden, wenn die Synchronisationskante mit FALSE_SIGNALED markiert wurde.

2.2.2.3 Ablaufhistorie

Der Kontrollfluss-Ausführungsgraph $CFS_{instance}$ beschreibt den aktuellen Zustand einer WF-Instanz. Aus ihm ist aber nicht ersichtlich, wie dieser Zustand entstanden ist. Dies ist aber z.B. für die dynamische Modifikation von WF-Instanzen relevant. Außerdem enthält der Ausführungsgraph keine Information darüber, welcher Benutzer eine Aktivität bearbeitet hat oder welcher WF-Server sie kontrolliert hat. Da solche Daten in dem hier betrachteten Zusammenhang benötigt werden, wird für jede WF-Instanz eine Ablaufhistorie verwaltet, in der alle relevanten, diese WF-Instanz betreffenden, Ereignisse eingetragen werden.

In Abb. 2.6 ist eine teilweise ausgeführte WF-Instanz und die sich daraus ergebende Ablaufhistorie (vereinfacht) dargestellt. Wie in [Rei00] formal beschrieben wurde, enthält eine Ablaufhistorie Einträge für das Starten und Beenden von Aktivitäteninstanzen. Da wir in der vorliegenden Arbeit die verteilte WF-Ausführung betrachten, wird in einem Eintrag zusätzlich der WF-Server vermerkt, der die jeweilige Aktivitäteninstanz kontrolliert hat. Der Aufbau und die Bedeutung der in einer Ablaufhistorie enthaltenen Einträge ist wie folgt[3]:

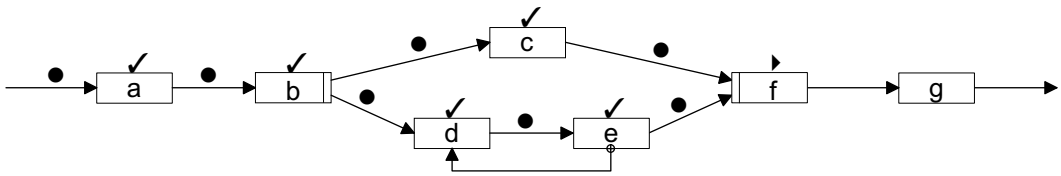
- START(Act, It, $Server$, $Actor$): Historieneintrag für den Start der It-ten Ausführung der Aktivität Act. Diese Aktivitäteninstanz wurde vom WF-Server $Server$ gesteuert und vom Benutzer $Actor$ bearbeitet.
- END(Act, It): Historieneintrag für die Beendigung der It-ten Ausführung der Aktivität Act.

Da in der Ablaufhistorie alle relevante Information über die Ausführung der Aktivitäteninstanzen gespeichert ist, kann aus ihr der aktuelle Zustand der WF-Instanz rekonstruiert werden.

2.2.3 Auf der Kontrollflussstruktur basierende Funktionen

Basierend auf dem Kontrollflussgraphen $CFS = (N, E, \ldots)$ eines WF-Typs bzw. auf dem Kontrollfluss-Ablaufgraphen $CFS_{instance} = (CFS, NS, ES, It)$ einer WF-Instanz können mehrere Funktionen definiert werden, die wir in dieser Arbeit benötigen werden. Da diese in [Zei99] bzw. in [Rei00] schon formal definiert wurden, wird hier lediglich ihre Bedeutung geklärt. Sie werden außerdem im Anhang A nochmals zusammengefasst.

[3] Außer den für diese Arbeit benötigten Feldern enthält ein Ablaufhistorieneintrag weitere Information, wie z.B. den Zeitpunkt des Startens bzw. Beendens der zugehörigen Aktivität.



... END(a, 1), START(b, 1, Server 1, U. Müller), END(b, 1), START(c, 1, Server 2, P. Adam), START(d, 1, Server 1, U. Müller), END(d, 1), START(e, 1, Server 2, F. Schmidt), END(e, 1), START(d, 2, Server 1, U. Müller), END(d, 2), START(e, 2, Server 2, J.I. Karlson), END(c, 1), END(e, 2), START(f, 1, Server 2, G. Schulz)

Abbildung 2.6 Teilweise ausgeführte WF-Instanz und die sich daraus ergebende Ablaufhistorie.

Da die in diesem Abschnitt definierten Funktionen Operationen durchführen, welche auf dem Kontrollflussgraphen $CFS = (N, E, \ldots)$ basieren, müsste ihnen dieser streng genommen immer als Parameter übergeben werden (z.B. $Succ_{Typ}^{CFS}(n)$). Wenn aber klar ist, welcher Kontrollflussgraph CFS gemeint ist (was im Folgenden stets zutrifft), verzichten wir auf diesen zusätzlichen Parameter.

2.2.3.1 Allgemeines

Zuerst wollen wir eine Funktion definieren, welche die Beziehung zwischen Aktivitäteninstanzen und Aktivitätentypen herstellt:

$\boldsymbol{n = ActTyp(a)}$**:** Die Funktion $ActTyp$ liefert zu einer Aktivitäteninstanz $a =< n, it >$ (vgl. Abschnitt 2.2.2.1) den zugehörigen Aktivitätentypen $n \in N$.

2.2.3.2 Nachfolgerbeziehungen

Die nachfolgend definierten Funktionen beziehen sich auf die Reihenfolge der Aktivitäten in der WF-Vorlage:

$\boldsymbol{M = Succ_{Typ}(n)}$**:** Die Nachfolgerfunktion $Succ$ berechnet für die Aktivität $n \in N$ die Menge der direkten Nachfolgeraktivitäten $M \subseteq N$ bezüglich der Kanten mit Kantentyp $E \in Typ$ ($Typ \subseteq \{CONTROL\_E, SYNC\_E, LOOP\_E\}$) im Kontrollflussgraphen $CFS = (N, E, \ldots)$. Die Menge M enthält also diejenigen Aktivitäten $n' \in N$, für die es eine Kante $e = (n, n', edgeType) \in E$ mit $edgeType \in Typ$ gibt.

$\boldsymbol{M = Succ^*_{Typ}(n)}$**:** Die Funktion $Succ^*$ berechnet die transitive Hülle von $Succ$. Die Ergebnismenge M enthält also auch die indirekten Nachfolgeraktivitäten von n bzgl. der in Typ spezifizierten Kantentypen. Die Aktivität n selbst ist nicht automatisch in M enthalten.

$\boldsymbol{M = Pred_{Typ}(n)}$**:** Diese Funktion ist ähnlich zur Funktion $Succ$ definiert, allerdings mit dem Unterschied, dass mit der Menge M die direkten Vorgängeraktivitäten von n bzgl. der angegebenen Kantentypen berechnet werden.

$\boldsymbol{M = Pred^*_{Typ}(n)}$**:** Diese Funktion berechnet die transitive Hülle von $Pred$, also zusätzlich die indirekten Vorgängeraktivitäten bzgl. der angegebenen Kantentypen.

Die nachfolgend beschriebenen Funktionen beziehen sich auf eine bestimmte WF-Instanz, weshalb ihnen streng genommen als zusätzlicher Parameter der Kontrollfluss-Ablaufgraph $CFS_{instance} = (CFS, NS, ES, It)$ übergeben werden müsste:

$B = SuccInst_{Typ}(a)$: Die von dieser Funktion berechnete Menge B enthält die direkten Nachfolgeraktivitäteninstanzen der Aktivitäteninstanz $a =< n, it >$ bzgl. Kanten der angegebenen Kantentypen Typ. Das heißt, zur Aktivierung einer Aktivität $b \in B$ musste eine von a nach b verlaufende Kante signalisiert werden, deren Kantentyp $edgeType$ in der Menge Typ enthalten ist.

$B = SuccInst^{*}_{Typ}(a)$: Diese Funktion berechnet die transitive Hülle von $SuccInst$, also zusätzlich die indirekten Nachfolgeraktivitäteninstanzen von a.

$B = PredInst_{Typ}(a)$: Mit $PredInst$ wird die Menge der direkten Vorgängeraktivitäteninstanzen von der Aktivitäteninstanz a berechnet.

$B = PredInst^{*}_{Typ}(a)$: Diese Funktion berechnet alle (indirekten) Vorgängeraktivitäteninstanzen von a.

2.2.3.3 Blockbezogene Funktionen

In ADEPT$_{base}$ hat jeder Block genau eine Start- und eine Endeaktivität (vgl. Abschnitt 2.2.1). Deshalb ist es möglich, für Verzweigungs- bzw. Schleifenblöcke die zur Startaktivität gehörende Endeaktivität anzugeben (und umgekehrt). Diese werden durch die nun beschriebenen Funktionen ermittelt:

$n' = Join(n)$: Die Funktion $Join$ berechnet für eine Split-Aktivität $n \in N$ (Startaktivität einer Verzweigung) die zugehörige Join-Aktivität $n' \in N$ (Endeaktivität der Verzweigung).

$n' = Split(n)$: $Split$ ist die zu $Join$ inverse Funktion: Sie berechnet die zur Join-Aktivität n gehörende Split-Aktivität n'.

$n' = Endloop(n)$: Die Funktion $Endloop$ berechnet die zur Schleifenstartaktivität n gehörende Schleifenendeaktivität n'.

$n' = Startloop(n)$: Diese Funktion berechnet die zur Schleifenendeaktivität n gehörende Schleifenstartaktivität n'.

2.2.3.4 Funktionen auf Ablaufhistorien

Im Folgenden werden einige Funktionen definiert, die auf Ablaufhistorien operieren:

$a = ActivityInstance(E)$: Die Funktion $ActivityInstance$ liefert zu einem Ablaufhistorieneintrag $E = Start(a, \ldots)$ bzw. $E = End(a, \ldots)$ die zugehörige Aktivitäteninstanz a.

$E \in H$: Diese Funktion bildet genau dann auf den Wert True ab, wenn der Historieneintrag E in der Ablaufhistorie H enthalten ist. Andernfalls ist das Ergebnis der Funktion False und es gilt $E \notin H$.

$E = LastEntry(H)$: Die Funktion $LastEntry$ berechnet den hintersten Eintrag E einer Ablaufhistorie H. Ist die Historie H leer ($H = \emptyset$), so wird der Wert $NULL$ zurückgegeben.

$H' = Append(H, E)$: Diese Funktion hängt den Eintrag E (hinten) an die Ablaufhistorie H an. Das Ergebnis ist wiederum eine Historie.

$H' = H \cap A$: Die Schnittmenge zwischen einer Historie H und einer Menge A von Aktivitäteninstanzen wird ermittelt, indem die Ablaufhistorie H auf diejenigen Einträge $E = Start(a, \ldots)$ bzw. $E = End(a, \ldots)$ reduziert wird, deren zugehörige Aktivitäteninstanz a in A enthalten ist ($a \in A$). Die Reihenfolge der Einträge in H' ist dieselbe wie in H. Das Funktionsergebnis H' ist erneut eine Ablaufhistorie.

$H' = H - A$: Die Differenz zwischen einer Historie H und einer Menge A von Aktivitäteninstanzen ist analog zum Schnitt definiert, wobei aber diejenigen Historieneinträge $E \in H$ in die Ergebnishistorie H' übernommen werden, deren zugehörige Aktivitäteninstanzen nicht in A enthalten sind.

2.3 Datenfluss

Bis jetzt wurden in diesem Kapitel ausschließlich Aspekte betrachtet, die den Kontrollfluss betreffen, also die Reihenfolgebeziehungen zwischen den Aktivitäten. Dieser Abschnitt widmet sich nun dem Datenfluss, welcher festlegt, welche Daten von welchen Aktivitäten gelesen bzw. geschrieben werden. Wir untersuchen zuerst die generell zur Verfügung stehenden Alternativen und wählen eine geeignete aus. Anschließend wird beschrieben, wie diese im Basismodell von ADEPT umgesetzt wurde.

2.3.1 Varianten zur Realisierung des Datenflusses

Das für die Realisierung des Datenflusses verwendete Modell hat Auswirkungen auf den Umfang der zu kommunizierenden Daten. So können mehrfach gespeicherte Daten unter Umständen mehrfach übertragen werden, wenn eine WF-Instanz auf einen anderen WF-Server migriert wird. Außerdem kann es vorkommen, dass Daten zu einer Aktivität transportiert werden, die überhaupt nicht benötigt werden. Das Datenmodell hat auch noch andere Auswirkungen auf ein WfMS, z.B. auf die Modellierung. Auf diese Aspekte wird hier aber nur am Rande eingegangen, da sie keinen Einfluss auf die Effizienz der WF-Steuerung haben.

Bei den im Folgenden betrachteten Varianten zur Realisierung des Datenflusses gibt es prinzipiell immer zwei Möglichkeiten, um Daten in einem WfMS zu verwalten. Eine Lösungsmöglichkeit ist, dass die Daten direkt vom WfMS gespeichert werden. Die Alternative ist, dass vom WfMS nur Referenzen auf die Daten verwaltet werden und die Daten selbst in einer externen Datenquelle (z.B. in einem Datenbank-Management-System (DBMS)) abgelegt sind. Die externe Speicherung von Daten kann u.a. die folgenden Gründe haben:

- Nicht in das WfMS integrierte Anwendungen müssen Zugriff auf diese Daten haben.
- Als Aktivitätenprogramm wird eine „Legacy Application“ verwendet, die so implementiert ist, dass sie sich ihre Daten aus einer bestimmten Datenbank holt (und sie nicht vom WfMS übergeben bekommt). Dies erfordert, dass die Daten in dieser externen Datenquelle belassen werden.
- Auch aus Datenschutzgründen kann es notwendig sein, dass dem WfMS nur Referenzen auf die eigentlichen Daten bekannt sind.

Man könnte nun annehmen, die Last sei immer geringer, wenn vom WfMS nur Referenzen auf Daten verwaltet werden, weil diese ein wesentlich kleineres Datenvolumen besitzen. Dass dem nicht so ist, soll durch das folgende Beispiel verdeutlicht werden: Von einem WF werde der vordere Teil in Denver bearbeitet und kontrolliert und der hintere Teil in Stuttgart. Die Bearbeitung findet also weiträumig verteilt statt. Wenn im vorderen Teil Daten geschrieben werden, so befinden sich diese auf in Denver lokalisierten Laufwerken. Werden diese Daten, auf welche das WfMS nur Referenzen verwaltet, nun von mehreren Aktivitäten des hinteren Teils des WF benötigt, so müssen sie mehrfach (bei jedem Zugriff) von Denver nach Stuttgart übertragen werden. Dies ist insbesondere für große (z.B. multimediale) Daten sehr teuer. Werden die Daten hingegen vom WfMS verwaltet, so müssen die nur einmal, bei der Migration, nach Stuttgart übertragen werden. Ab diesem Zeitpunkt kann lokal

auf sie zugegriffen werden. Durch die Verwendung von Referenzen auf Daten reduziert sich also die Belastung des WfMS, nicht aber die des Kommunikationssystems. Wie dieses Beispiel zeigt, ist es generell günstiger – falls dies möglich ist – Daten vom WfMS verwalten zu lassen, als sie extern zu speichern. Nur dann kann das bestehende Optimierungspotential voll ausgenutzt werden. Es bleibt aber die Aufgabe des WF-Designers, für jedes Datenelement zu entscheiden, ob es vom WfMS verwaltet werden soll, oder ob nur eine Referenz gespeichert werden soll.

2.3.1.1 Objektmigrationsansatz

Elektronische Umlaufmappen (Electronic Circulation Folders) [iABa, KRW90, KR91a] werden vor allem in dokumentenorientierten WfMS wie ProMInanD [KR91b, Kar94] verwendet. Die komplette WF-Instanz wird jeweils zu demjenigen Knoten transportiert, auf welchem die aktuelle Aktivität ausgeführt werden soll. Eine Elektronische Umlaufmappe enthält außer den Daten, welche die Anwendungen benötigen (meist Dokumente), oft auch noch die komplette Schemainformation des Prozesses.

Da die WF-Instanz dem Bearbeiter einer Aktivität immer komplett zur Verfügung steht, ermöglicht dieses Modell eine hohe Flexibilität. So können von einer Aktivität auch Dokumente verwendet werden, die für diesen Schritt eigentlich nicht vorgesehen waren. Außerdem ist die komplette Prozessbeschreibung bei jeder WF-Instanz verfügbar, so dass der Ablauf dieser WF-Instanz dynamisch verändert werden kann [KR90a, VE92]. Diesen Vorteilen stehen jedoch hohe Kommunikationskosten und damit auch eine hohe Belastung für die WF-Server gegenüber. Sie entstehen dadurch, dass auch nicht benötigte Dokumente und die von den Anwendungen nicht verwendete Ablaufbeschreibung vom WF-Server zu den Clients und zurück transportiert werden.

Schwierigkeiten macht beim Objektmigrationsansatz die Ausführung paralleler Zweige. Wird dieselbe Elektronische Umlaufmappe in mehreren Zweigen verwendet, so entsteht ein Problem am Synchronisationspunkt. Für Dokumente, die in mehreren Zweigen verändert wurden, ist nicht entscheidbar, welche Version verwendet werden soll. Dies kann zum Verlust von Änderungen (Lost Updates) führen.

2.3.1.2 Input-/Output-Container

In FlowMark [IBM96b, LA94, LR94, LR00] werden die Ein- und Ausgabeparameter von Aktivitäten aufeinander abgebildet, d.h. für jedes Eingabedatum einer Aktivität muss die Aktivität und derjenige Ausgabeparameter angegeben werden, dessen Wert übernommen werden soll[4]. Auf diese Weise wird der Datenfluss explizit modelliert. Um nun in einen Prozessschritt die Daten von schon abgeschlossenen Aktivitäten verwenden zu können, werden die Eingabecontainer aller Teilschritte physisch gespeichert. Die Daten sind also redundant vorhanden, falls sie von mehreren Aktivitäten benötigt werden.

Der Gültigkeitsbereich von Variablen ist bei diesem Modell immer auf eine Aktivität beschränkt, ihr Inhalt kann lediglich nach Beendigung des Aktivitätenprogramms anderen Variablen zukünftiger Aktivitäten übergeben werden. Da deshalb zwei Aktivitäten nie eine gemeinsame Variable verwenden, können Synchronisationsprobleme nur dann auftreten, wenn der Datenfluss explizit so modelliert wurde, dass eine Variable ihren Inhalt von mehreren Aktivitäten bezieht. In diesem (leicht erkennbaren) Fall ist es die Aufgabe des WF-Designers, die gewünschte Synchronisation sicherzustellen. Ein

[4]Es ist auch möglich hier mehrere Aktivitäten anzugeben. Dies ist notwendig, um bei bedingten Verzweigungen die Daten von dem jeweils durchlaufenen Zweig übernehmen zu können. Werden mehrere Zweige parallel durchlaufen, so kann dies allerdings zu Lost Updates führen.

Nachteil dieses Ansatzes ist, dass das Modellieren des kompletten Datenflusses sehr aufwendig ist. Des Weiteren kann es notwendig werden, von einem Verzweigungsprädikat benötigte Daten durch eine Aktivität (d.h. durch das Anwendungsprogramm) durchzuschleusen, da für Verzweigungsprädikate nur die Daten aus dem Ausgabecontainer der unmittelbaren Vorgängeraktivität gültig sind.

Wird eine WF-Instanz von nur einem einzigen WF-Server kontrolliert, so sind Input-/Output-Container sehr effizient, da für die Bearbeitung einer Aktivität nur die Daten zu einem Client transportiert werden müssen, die (der Modellierung zufolge) wirklich benötigt werden. Die Daten sind lediglich redundant in der WF-DB vorhanden, was keine großen Kosten verursacht. Anders sieht dies aus, falls eine komplette WF-Instanz zu einem anderen Server migriert werden soll. Dann müssen die mehrfach vorhandenen Daten zum neuen Server kopiert werden, was sowohl den Server als auch das Netzwerk belastet. Eine Ausnahme bilden Subprozesse mit eigenem Input-Container, die von einem entfernten WF-Server gesteuert werden sollen. Zu ihnen müssen nur die wirklich benötigten Daten dieser einen (zusammengesetzten) Aktivität transportiert werden.

2.3.1.3 Prozessglobale Variablen

Das Konzept der prozessglobalen Variablen wird z.B. von WorkParty [Rup95, Sie95a, Sie95b] verwendet. Für einen Prozess können Variablen definiert werden (ähnlich wie die Variablen einer Prozedur in einer Programmiersprache). Für jede Aktivität kann nun angegeben werden, welche dieser prozessglobalen Variablen (im Folgenden auch Datenelemente genannt) sie liest bzw. schreibt.

Alle Datenelemente sind nur einfach vorhanden und zu einer Anwendung werden nur solche Daten transportiert, die wirklich benötigt werden. Bezüglich der Kommunikationskosten ist dieser Ansatz optimal. Deshalb wurde er für das ADEPT-Basismodell ausgewählt. Die Details seiner Umsetzung werden im nachfolgenden Abschnitt beschrieben.

Ein Nachteil der Verwendung von prozessglobalen Variablen ist, dass beim Zugriff durch Aktivitäten paralleler Zweige auf dasselbe Datenelement Zugriffskonflikte und Lost Updates auftreten. Diese lassen sich aber z.B. dadurch verhindern, dass jedes Datenelement nur in einem der parallelen Zweige geschrieben werden kann und in den anderen als Read-only Parameter modelliert werden muss. Wie im nächsten Abschnitt beschrieben wird, wurde für ADEPT$_{base}$ ein solches Vorgehen gewählt.

2.3.2 Datenfluss im ADEPT-Basismodell

Nachdem die prinzipiellen Möglichkeiten zu Realisierung des Datenflusses in einem WfMS erörtert wurden, wird nun das von ADEPT verwendete Verfahren näher erläutert. Dazu wird zuerst beschrieben, wie der Datenfluss in ADEPT$_{base}$ definiert wird, und anschließend wird erläutert, welche Regeln dabei eingehalten werden müssen, um die Konsistenz des Datenflusses garantieren zu können.

2.3.2.1 Realisierung des Datenflusses

In ADEPT$_{base}$ wird der Datenfluss explizit modelliert, indem für jede Aktivität die Ein- und Ausgabedaten festgelegt werden. Dies erfolgt durch die Definition von Lese- und Schreibkanten (auch Datenflusskanten genannt), welche die betroffene Aktivität und das Datenelement verbinden. Ein Beispiel für eine Lesekante ist in Abb. 2.7 dargestellt: das Datenelement d_1 ist durch eine Datenflusskante (in dieser Arbeit gestrichelt gezeichnet) mit der Aktivität d verbunden. Die von Aktivität a zu d_1 führende Datenflusskante repräsentiert einen Schreibzugriff (Schreibkante).

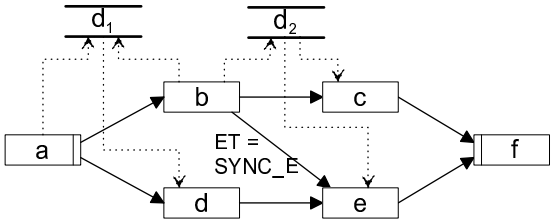


Abbildung 2.7 Beispiel für einen Kontrollflussgraphen inklusive Datenflusskanten.

Wir schon erwähnt, bezeichnen wir prozessglobale Variablen auch als Datenelemente. Von diesen Datenelementen werden in ADEPT Versionen verwaltet, d.h. dem WfMS sind auch alle alten Werte eines Datenelements bekannt (also die vollständige Historie des Datenelements). Diese Datenhistorie wird für das teilweise Zurücksetzen einer WF-Instanz benötigt. Um die verschiedenen Versionen eines Datenelementes unterscheidbar zu machen, wird in der Datenhistorie für ein Datenelement nicht nur dessen Wert verwaltet, sondern zusätzlich die Identifikation (ID) der Aktivitäteninstanz, welche diesen Wert geschrieben hat. Damit ergeben sich Datenhistorieneinträge mit folgendem Aufbau:
$<$ *Datenelement, Wert, SchreibendeAktivitäteninstanz* $>$

Da i.d.R. mehrere Versionen eines Datenelements existieren, stellt sich nun die Frage, auf welche Version beim Lesen zugegriffen wird. Dies ist nicht die zeitlich zuletzt geschriebene Version, sondern diejenige, die von der letzten Vorgängeraktivität bzgl. Kontroll-, Synchronisations- und Schleifenkanten geschrieben wurde. Es werden also keine Werte von Datenelementen gelesen, die von Aktivitäten paralleler Zweige geschrieben wurden, es sei denn, es existiert eine entsprechende Synchronisationskante zwischen diesen Zweigen. In dem Beispiel aus Abb. 2.7 liest die Aktivität d also den von der Aktivität a geschriebenen Wert des Datenelements d_1, selbst wenn die Aktivität b zeitlich davor einen anderen Wert von d_1 geschrieben hat. Da wegen der Synchronisationskante $b \rightarrow e$ die Aktivität e ein Nachfolger von b bzgl. des Kontrollflusses ist, liest e den von b geschriebenen Wert des Datenelements d_2. Der Datenfluss erfolgt also stets „entlang dem Kontrollfluss“, was zu leicht verständlichen WF-Vorlagen führt.

2.3.2.2 Datenfluss bei Subprozessen

Zur Strukturierung komplexer Abläufe und zur einfacheren Wiederverwendung von Prozessfragmenten ist es in ADEPT$_{base}$ möglich, bei der Modellierung Subprozesse zu verwenden. Im übergeordneten WF wird ein Subprozess durch eine einzige Aktivität repräsentiert. Um den vertikalen Datenfluss zu einem Sub-WF zu realisieren, hat diese (zusammengesetzte) Aktivität dieselben Ein- und Ausgabeparameter wie der Subprozess, der damit auf die entsprechenden Daten zugreifen kann. Die Eingabeparameter eines Subprozesses werden logisch als Ausgabeparameter der (leeren) Startaktivität ($NT = STARTFLOW$) betrachtet. Damit stehen diese Daten allen anderen Aktivitäten des Subprozesses zur Verfügung. Analog werden die Ausgabedaten des Subprozesses als Eingabedaten der leeren Endeaktivität angesehen. Der innerhalb des Subprozesses stattfindende horizontale Datenfluss ist wie in Abschnitt 2.3.2.1 beschrieben realisiert.

Subprozesse werden in dieser Arbeit nicht näher betrachtet, da sie nur zur Strukturierung bei der Modellierung dienen und für die effiziente verteilte WF-Steuerung nicht von Belang sind. Sie bilden, ähnlich wie z.B. eine Verzweigung, lediglich einen Block im ADEPT-Modell. Deshalb wird eine Aktivität, die einen Sub-WF repräsentiert, durch diesen ersetzt. Wir erhalten so eine „flache WF-Vorlage“, die von den in dieser Arbeit beschriebenen Algorithmen verwendet wird. Auf diese Weise ersparen

wir uns jeweils die Behandlung des Sonderfalls, dass eine betrachtete Aktivität selbst aus mehreren Aktivitäten besteht, die ebenfalls betrachtet werden müssen. In Abschnitt 3.4.2 wird diskutiert, warum das Konstrukt des Subprozesses wenig geeignet ist, um eine verteilte WF-Ausführung zu realisieren.

2.3.2.3 Konsistenz des Datenflusses

In ADEPT ist es ein Ziel, die Konsistenz des Datenflusses zu garantieren. So soll möglichst schon bei der Modellierung eines Prozesses ausgeschlossen werden, dass noch nicht existierende Datenelemente als Eingabeparameter von Aktivitäten oder für Verzweigungs- und Schleifenbedingungen verwendet werden, da dies zu Laufzeitfehlern führen würde. Ebenso soll das nebenläufige Schreiben eines Datenelements durch parallel ausgeführte Aktivitäten verhindert werden, um Lost Updates zu vermeiden.

Die Konsistenz des Datenflusses wird durch Strukturierungsregeln sichergestellt. Verstößt eine WF-Vorlage gegen eine dieser Strukturierungsregeln, so wird der Modellierer gewarnt. Normalerweise sollte er einen solchen Prozess nicht freigeben. Aufgrund von zusätzlichem semantischen Wissen (z.B. über das Zusammenspiel von Verzweigungsbedingungen) kann er dies aber dennoch tun, falls er sicher ist, dass die Ausführung von WF-Instanzen dieses Typs unproblematisch ist.

Im Folgenden werden die Strukturierungsregeln von ADEPT$_{base}$ vorgestellt und erläutert:

- Die erste Regel erzwingt, dass ein Datenelement mit Sicherheit geschrieben wurde, bevor es gelesen wird. Das bedeutet, egal welchen Weg (bei bedingten Verzweigungen) die Ausführung einer WF-Instanz nimmt, muss jedes Datenelement beim Lesen immer einen gültigen Wert haben, also von einer Vorgängeraktivität (bzgl. des Kontrollflusses) geschrieben worden sein.
- Die zweite Regel verbietet, dass ein Datenelement von mehreren Aktivitäten geschrieben wird, die in keiner Kontrollfluss-Reihenfolge zueinander stehen. Das heißt, Aktivitäten, die zu parallelen Zweigen gehören und zwischen denen es keine durch Synchronisationskanten erzwungene Reihenfolgebeziehung gibt, dürfen nicht ein und dasselbe Datenelement schreiben. Wäre dies erlaubt, so würden beim Zusammenführen der parallelen Zweige Lost Updates entstehen, da nur einer der Werte übernommen werden kann.[5] Diese Strukturierungsregel sorgt außerdem dafür, dass stets eindeutig ist, welche Version beim Lesen eines Datenelements verwendet werden muss, da alle Schreibzugriffe auf das Datenelement in einer Reihenfolgebeziehung stehen, so dass der zuletzt von einer Vorgängeraktivität geschriebene Wert eindeutig identifiziert werden kann.

2.3.2.4 Funktionen mit Bezug zum Datenfluss

In diesem Abschnitt werden einige Funktionen definiert, die sich auf den Datenflussgraphen beziehen. Ebenso wie die auf dem Kontrollfluss basierenden Funktionen werden sie im Anhang A nochmals zusammengefasst.

$D = ReadSet(n)$: Die von dieser Funktion berechnete Menge D enthält diejenigen Datenelemente, die von der Aktivitäteninstanz $a = < n, it >$ gelesen werden. Dies sind also die für den Aktivitätentyp n festgelegten Eingabeparameter.

[5] Lost Updates könnten auch verhindert werden, indem der Zugriff auf ein Datenelement durch Aktivitäten paralleler Zweige explizit synchronisiert wird. Dann müsste aber bei verteilter WF-Ausführung bei jedem Zugriff auf ein Datenelement kommuniziert werden, was sehr große Kosten verursachen würde. Deshalb ist die in ADEPT$_{base}$ realisierte Semantik für den Zugriff auf Datenelemente essentiell für eine effiziente verteilte WF-Steuerung.

$A = Reader(d)$: Die Funktion $Reader$ berechnet die Menge A der Aktivitäteninstanzen, welche das Datenelement d bei der fraglichen WF-Instanz gelesen haben.

$b = LastWriter(d, a)$: $LastWriter$ berechnet die letzte Vorgängeraktivitäteninstanz b der Aktivitäteninstanz a, welche das Datenelement d geschrieben hat. Wenn die Aktivitäteninstanz a das Datenelement d lesen möchte, so wird also die von b geschriebene Version von d gelesen.

2.4 Organisationsmodell und Bearbeiterzuordnungen

Die Bearbeiterzuordnungen legen für alle Aktivitäten $n \in N$ eines Kontrollflussgraphen $CFS = (N, E, \ldots)$ die Menge der Benutzer fest, die diese Aktivität bearbeiten dürfen (im Folgenden „qualifizierte Bearbeiter“ genannt). Dabei ist $BearbZuordn_n$ ein Ausdruck, der zur Ausführungszeit der WF-Instanz ausgewertet wird. Die Bearbeiterzuordnung einer Aktivität ist in den in dieser Arbeit verwendeten Abbildungen jeweils oberhalb der zugehörigen Aktivität angegeben. Da sich Bearbeiterzuordnungsausdrücke auf das Organisationsmodell des WfMS beziehen können, wird im Folgenden zuerst beschrieben, welche Informationen dieses bei ADEPT beinhaltet. Anschließend wird auf die Bearbeiterzuordnung detailliert eingegangen.

2.4.1 Organisationsmodell

In einem Organisationsmodell eines WfMS wird, ähnlich wie bei einem Geschäftsprozess-Modellierungswerkzeug [Sch96, Sch98, VB96], die Organisationsstruktur des betroffenen Unternehmens beschrieben – zumindest für denjenigen Teil der Organisation, der in das WfMS involviert ist. Im Folgenden wird beschrieben, welche Informationen ein Organisationsmodell im Kontext dieser Arbeit mindestens enthalten muss. Diese Informationen werden insbesondere für die Auswertung von „abhängigen Bearbeiterzuordnungen“ (siehe Abschnitt 2.4.3) benötigt.

Im Zusammenhang mit Bearbeiterzuordnungen sind natürlich die Benutzer des WfMS von besonderem Interesse, da sie die Bearbeiter der Aktivitäten darstellen. Deshalb bilden sie die zentrale Entität des Organisationsmodells. Für jeden Benutzer werden die Rollen gespeichert, die er innehaben kann, und die Organisationseinheit (OE), der er angehört. Außerdem wird weitere benutzerspezifische Information, wie Kompetenzen, Berechtigungen oder Stellvertreterregelungen verwaltet (siehe [Kub98]). Dies ist im Kontext der vorliegenden Arbeit aber nicht von besonderem Interesse. All diese Information kann verwendet werden, um in den Bearbeiterzuordnungen der Aktivitäten diejenigen Benutzer zu spezifizieren, welche die entsprechende Aktivität ausführen dürfen.

Das Organisationsmodell verwaltet für jeden Benutzer außerdem diejenigen Domains, in denen er sich anmelden darf. Dies können mehrere sein, da z.B. ein Arzt, je nach gerade eingenommener Rolle, dem Domain seiner Krankenstation oder dem der Notaufnahme angehören kann. Welchem Domain ein beim WfMS aktuell angemeldeter Benutzer tatsächlich angehört, wird als Laufzeitinformation ebenfalls vom Organisationsmodell verwaltet. Dasselbe gilt für aktuell eingenommene Rollen und sonstige Eigenschaften des Benutzers. Diese Laufzeitinformationen haben aber keinen Einfluss auf die qualifizierten Bearbeiter einer Aktivität, da auch aktuell nicht angemeldete Benutzer als qualifizierte Bearbeiter der Aktivität betrachtet werden. Der Grund dafür ist, dass sie die Aktivität bearbeiten dürfen, sobald sie sich beim WfMS anmelden (auch wenn die Aktivität schon zuvor aktiviert wurde).

2.4.2 Unabhängige Bearbeiterzuordnungen

Die Bearbeiterzuordnung $BearbZuordn_n$ definiert die Menge der qualifizierten Bearbeiter der Aktivität n. Dieser Ausdruck kann Daten aus dem Organisationsmodell referenzieren, bei unabhängigen Bearbeiterzuordnungen aber keine Laufzeitdaten der WF-Instanz. Deshalb qualifiziert sich bei der Ausführung jeder Instanz der Aktivität n dieselbe Bearbeitermenge. Diese ändert sich lediglich, wenn das Organisationsmodell verändert wird.

Der einfachste Fall einer Bearbeiterzuordnung ist die direkte Zuordnung eines Benutzers, z.B.:
$BearbZuordn_n =$ `Klaus Meier`
Diese Art der Bearbeiterzuordnung ist aber sehr problematisch, da sie bei einer Veränderung des Organisationsmodells angepasst werden muss. Das führt dazu, dass, wenn der entsprechende Mitarbeiter das Unternehmen verlässt, die WF-Vorlage verändert werden muss. Da diese Vorgehensweise sehr aufwendig und fehleranfällig ist, sollte sie nicht verwendet werden. Stattdessen sollten Bearbeiter stets über Rollen zugeordnet werden:
$BearbZuordn_n =$ `Rolle = Arzt`
Wird bei dieser Bearbeiterzuordnung ein Mitarbeiter durch einen anderen ersetzt, dem natürlich dieselbe Rolle zugeordnet wird, so übernimmt dieser automatisch dessen Aufgaben. Da die WF-Vorlage unverändert bleibt, sind die Änderungen auf das Organisationsmodell beschränkt. Häufig genügt die Festlegung einer Rolle jedoch nicht für die Spezifikation der Bearbeiter einer Aktivität. So soll eine bestimmte Untersuchung eben nicht von einem beliebigen Arzt durchgeführt werden, sondern von einem Arzt der Abteilung Radiologie. Dann muss in der Bearbeiterzuordnung zusätzlich zur Rolle die OE des Bearbeiters spezifiziert werden:
$BearbZuordn_n =$ `Rolle = Arzt` $\wedge$ `Abteilung = Radiologie`
Ebenso ist auch die Verwendung der im vorherigen Abschnitt erwähnten Kompetenzen, Berechtigungen, o.ä. in den Bearbeiterzuordnungen möglich. Dadurch können Bearbeiterzuordnungen entstehen, die komplexe logische Ausdrücke beinhalten.

2.4.3 Abhängige Bearbeiterzuordnungen

Abhängige Bearbeiterzuordnungen sind in der Praxis unbedingt erforderlich, um die zur Modellierung realer Abläufe benötigte Ausdrucksmächtigkeit zu erhalten [BF99]. Das zeigt sich auch daran, dass sie von kommerziellen Systemen wie IBM FlowMark [IBM96b, LA94, LR94]) angeboten werden. Deshalb werden die entsprechenden Ausdrücke in diesem Abschnitt eingeführt und es wird erläutert, wie bei deren Verwendung die Menge der potentiell qualifizierten Bearbeiter einer Aktivität zur Modellierungszeit ermittelt werden kann.

2.4.3.1 Verwendung von abhängigen Bearbeiterzuordnungen

In vielen Anwendungen sind für bestimmte Aktivitäten zwar prinzipiell Mitarbeiter aus verschiedenen OE zuständig, für eine bestimmte WF-Instanz sind aber doch nur Mitarbeiter aus einer bestimmten OE zuständig. Ein solches Szenario haben wir schon in Abschnitt 1.2.2 kennengelernt; Ein ähnliches Beispiel aus dem Krankenhausbereich ist in Abb. 2.8 dargestellt (siehe auch [BD99a, BD00a]). Ein Patient wird in die Ambulanz eingeliefert und danach von einer Pflegekraft einer a priori unbekannten Station aufgenommen (Aktivität 1). Ab diesem Zeitpunkt ist nur noch Personal dieser Station für den Patienten zuständig (Aktivitäten 2-4). So wird er von einem Arzt der entsprechenden Krankenstation

untersucht und derselbe Arzt schreibt anschließend einen Kurzbefund. Auch die Aktivität „Patient vorbereiten“ wird von einer Pflegekraft derselben Station durchgeführt.

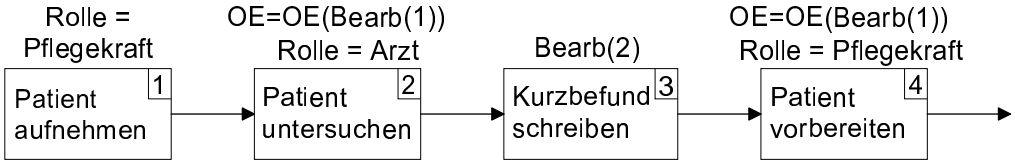


Abbildung 2.8 Beispiel für einen WF mit abhängigen Bearbeiterzuordnungen.

Damit eine abhängige Bearbeiterzuordnung der Aktivität n zur Ausführungszeit überhaupt aufgelöst werden kann, dürfen nur Aktivitäten m referenziert werden, die vor Aktivität n ausgeführt werden (Vorgängeraktivitäten). Die folgenden Typen von abhängigen Bearbeiterzuordnungen werden in dieser Arbeit berücksichtigt:

1. $BearbZuordn_n =$ `Bearb(m)`
   Aktivität n soll vom selben Bearbeiter ausgeführt werden wie Aktivität m
2. $BearbZuordn_n =$ `OE = OE(Bearb(m))` $\wedge$ `...`
   Der Bearbeiter der Aktivität n soll derselben OE angehören, wie der Bearbeiter von Aktivität m. Außerdem soll er die weiteren angegebenen Kriterien erfüllen.
3. Manchmal werden auch Bearbeiterzuordnungen benötigt, die von WF-Daten abhängig sind. Ein Beispiel hierfür ist eine Aktivität, bei der die ausführende OE von der dabei durchzuführenden Untersuchung abhängig ist:
   $BearbZuordn_n =$ `(Untersuchung = Röntgen: OE = Radiologie` $\wedge$ `Rolle = MTA)`
   $\vee$ `(Untersuchung = Endoskopie: OE = Chirurgie` $\wedge$ `Rolle = Arzt)`
   Allgemein haben solche Bearbeiterzuordnungen die Form:
   $BearbZuordn_n = Bedingung^1$`:` $BearbZuordn^1$ $\vee$
   $Bedingung^2$`:` $BearbZuordn^2$ $\vee$ ...

Außer diesen Typen von Bearbeiterzuordnungen sind auch noch weitere denkbar, auf die in dieser Arbeit aber nicht detailliert eingegangen wird. So kann z.B. die folgende Bearbeiterzuordnung verwendet werden, um ein 4-Augen-Prinzip zu realisieren:
$BearbZuordn_n =$ `Bearb` $\neq$ `Bearb(m)` [$\wedge$ `OE = OE(Bearb(m))`] $\wedge$ ...
Die Bearbeiter der Aktivitäten m und n sollen nicht identisch sein [aber derselben OE angehören]. Die Auswahl des Bearbeiters von Aktivität n kann noch durch weitere Kriterien eingeschränkt werden.
Soll die Aktivität n vom Vorgesetzten des Bearbeiters der Aktivität m ausgeführt werden, so ergibt sich die folgende Bearbeiterzuordnung:
$BearbZuordn_n =$ `Vorgesetzter(Bearb(m))`
Je nach Anwendung sind auch noch weitere Typen von Bearbeiterzuordnungen denkbar.

2.4.3.2 Potentielle Bearbeiter einer Aktivität

Bei unabhängigen Bearbeiterzuordnungen (z.B. $BearbZuordn_n =$ `Rolle = Arzt`) kann die Menge der qualifizierten Bearbeiter von Aktivität n schon zur Modellierungszeit aus $BearbZuordn_n$ ermittelt werden, indem für alle Benutzer des WfMS geprüft wird, ob sie die entsprechende Bedingung $BearbZuordn_n$ erfüllen. Dies ist bei abhängigen Bearbeiterzuordnungen nicht direkt möglich. So hat die Aktivität 3 in Abb. 2.8 die $BearbZuordn_3 =$ `Bearb(2)`, woraus nicht direkt auf die potentiellen Bearbeiter der Aktivität 3 geschlossen werden kann.

Da die Menge der Bearbeiter, die sich potentiell für die Ausführung einer Aktivität qualifizieren, von zahlreichen in dieser Arbeit vorgestellten Algorithmen benötigt wird, wird nun ein Verfahren vorgestellt, um diese Menge $PotBearb_n$ zu berechnen. $PotBearb_n$ enthält also diejenigen Bearbeiter, welche für die Bearbeitung der Aktivität n theoretisch in Frage kommen. Diese Menge ist unabhängig von einer konkreten WF-Instanz und wird schon zur Modellierungszeit berechnet. Die Menge der Bearbeiter, welche sich für eine konkrete Aktivitäteninstanz zur Ausführungszeit dann tatsächlich qualifizieren, ist damit eine Teilmenge von $PotBearb_n$. Um $PotBearb_n$ zu berechnen, durchläuft der Algorithmus 2.1 sukzessive alle Aktivitäten n der WF-Vorlage.[6] Wenn er dabei auf eine Aktivität stößt, deren Bearbeiterzuordnung von einer anderen Aktivität m abhängig ist, so verwendet er $PotBearb_m$ (die Menge der potentiellen Bearbeiter der referenzierten Aktivität m), um die Menge der potentiellen Bearbeiter der referenzierenden Aktivität n zu berechnen. Wie dies funktioniert, soll am Beispiel $BearbZuordn_n =$ `OE = OE(Bearb(m)) ∧ Rolle = Arzt` erläutert werden: $PotBearb_n$ wird berechnet, indem der Algorithmus alle Bearbeiter $u_1 \in PotBearb_m$ der Aktivität m durchläuft und für jeden Benutzer u_2 prüft, ob er die Bearbeiterzuordnung von Aktivität n erfüllt, unter der Annahme, dass u_1 die Aktivität m bearbeitet hat. Dies ist in diesem Beispiel gegeben, wenn der Benutzer u_2 derselben OE angehört wie u_1 und ihm zusätzlich die Rolle Arzt zugeordnet ist. Die WF-Vorlage wird in partieller Ordnung[7] durchlaufen; außerdem können in Bearbeiterzuordnungen nur Vorgängeraktivitäten referenziert werden. Deshalb ist dem Algorithmus, wenn er $PotBearb_n$ berechnet, $PotBearb_m$ schon bekannt.

Algorithmus 2.1 (Berechnung von $PotBearb_n$ für Aktivität n)
input
$CFS = (N, E, \ldots)$: Kontrollflussgraph des betrachteten WF-Typs
$BearbZuordn_n$: $\forall n \in N$: Bearbeiterzuordnung der Aktivität n
output
$PotBearb_n$: $\forall n \in N$: potentiell qualifizierte Bearbeiter der Aktivität n
begin
for each Aktivität $n \in N$ (in partieller Ordnung bezüglich Kontrollfluss) **do**
$PotBearb_n = \emptyset$;
// abhängige Bearbeiterzuordnung vom Typ 1 oder 2:
if $BearbZuordn_n$ referenziert eine Aktivität m **then**
for each Benutzer $u_1 \in PotBearb_m$ **do**
for each Benutzer u_2 **do**
if u_2 erfüllt $BearbZuordn_n$, falls u_1 die Aktivität m bearbeitet hat **then**
$PotBearb_n = PotBearb_n \cup \{u_2\}$;
// unabhängige Bearbeiterzuordnung oder abhängige Bearbeiterzuordnung vom Typ 3:
else
for each Benutzer u **do**
if u erfüllt $BearbZuordn_n$ **then**
$PotBearb_n = PotBearb_n \cup \{u\}$;
end.

[6] In [BD00b] wird ein Algorithmus zur Berechnung von $PotBearb_n$ vorgestellt, der Ausdrücke ermittelt, welche die Menge der potentiellen Bearbeiter einer Aktivität beschreiben. Dieser Algorithmus ist etwas effizienter, aber sehr stark von den konkret betrachteten Bearbeiterzuordnungsausdrücken abhängig. Da $PotBearb_n$ nur ein einziges Mal (nach der Festlegung der Bearbeiterzuordnungen) ausgeführt werden muss, spielt die Effizienz dabei keine besondere Rolle, weshalb auf das optimierte Verfahren hier nicht weiter eingegangen wird.

[7] Partielle Ordnung bezüglich Kontrollfluss bedeutet, dass die Aktivität n_2 nicht vor der Aktivität n_1 bearbeitet wird, falls $n_1 \in Pred^*_{\{CONTROL\_E, SYNC\_E\}}(n_2)$.

2.5 Arbeitslistenverwaltung

Die Art und Weise, wie ein WfMS die Arbeitslisten der Benutzer aktualisiert, hat Einfluss auf die dadurch entstehenden Kommunikationskosten und damit auf die Effizienz des verteilten WF-Managements. Für das Aktualisieren der Arbeitslisten gibt es prinzipiell zwei Möglichkeiten: Entweder erfragt der WF-Client vom Server seine aktuelle Arbeitsliste (Polling) oder der Server benachrichtigt bei Änderungen die betroffenen Clients, ohne von außen angestoßen zu werden. Diese beiden Methoden werden im Folgenden diskutiert.

2.5.1 Polling

Beim Polling wendet sich der WF-Client an den Server, um die aktuelle Arbeitsliste des zugehörigen Benutzers zu erhalten. Dies kann auf die explizite Anforderung des Benutzers hin geschehen, in festen Zeitabständen oder ereignisorientiert (z.B. beim Beenden einer Aktivität). Dieses Verfahren wird z.B. vom System WorkParty [Sie95a, Sie95c] verwendet. Unter der Annahme, dass das Polling der Clients unabhängig voneinander erfolgt, z.B. in festen durchschnittlichen Zeitabständen, ist die durch das Aktualisieren der Arbeitslisten erzeugte Last proportional zur Anzahl der Benutzer des WfMS.

Das Polling führt zu Nachteilen, wenn das Verteilungsmodell des WfMS mehrere Server vorsieht. Ein Client muss dann seine Arbeitsliste bei allen WF-Servern erfragen, so dass jeder Server alle Clients bedienen muss, wodurch er zu einem Engpass werden kann. Eigentlich ist es für einen Client aber gar nicht notwendig, stets bei allen WF-Servern nach Änderungen seiner Arbeitsliste zu fragen, da sich die Mehrzahl der Server in fremden Geschäftsbereichen befinden und deshalb (fast) nie Aktivitäten des entsprechenden Benutzers kontrollieren. Allerdings ist es algorithmisch schwierig, diese Server zu identifizieren und dabei den Fall zu berücksichtigen, dass einer dieser Server ausnahmsweise doch einmal eine Aktivität für diesen Benutzer kontrolliert (z.B. bei Inkrafttreten einer Vertreterregelung oder bei einer dynamischen Änderung der Bearbeiterzuordnung).

2.5.2 Server aktualisiert die Arbeitslisten

Beim Aktualisieren der Arbeitslisten durch den Server ist die Aufrufrichtung umgekehrt, d.h. der WF-Server benachrichtigt seine Clients über Änderungen. Beim Polling kann es vorkommen, dass ein Client eine Aktualisierung seiner Arbeitsliste wünscht, obwohl sie sich seit der letzten Übertragung nicht geändert hat. Dies lässt sich nicht verhindern, da der Client nicht über die dazu notwendige Information verfügt. Im Gegensatz dazu verfügt der Server über diese Information. Er weiß, wann ein Eintrag in die Arbeitslisten eingefügt werden muss und wann er wieder entfernt werden kann. Diese Information kann folgendermaßen genutzt werden: Der Server sendet einem Client nur dann eine neue Arbeitsliste, wenn sich diese geändert hat. Mit dieser Methode aktualisiert z.B. das System ProMInanD [iABa, iABb, Kar94] seine Arbeitslisten. Bei diesem Verfahren ist es auch möglich, maximale Verzögerungszeiten für das Erscheinen von Einträgen in den Arbeitslisten der Benutzer zu garantieren. Das Aktualisieren der Arbeitslisten durch den Server ist also die geeignete Wahl, falls es Aktivitäten mit hohem Aktualitätsbedarf gibt.

Der Nachteil dieses Verfahrens ist, dass es zu einer sehr hohen Last führt, falls bei jeder Änderung sofort alle betroffenen Arbeitslisten aktualisiert werden. Die Häufigkeit der Änderung einer Arbeitsliste ist – im ungünstigsten Fall – proportional zur Anzahl der bearbeiteten Aktivitäteninstanzen und damit

zur Anzahl der Benutzer des WfMS. Da die Anzahl der existierenden Arbeitslisten ebenfalls proportional zu dieser Benutzeranzahl ist, ergibt sich für das Aktualisieren der Arbeitslisten ein Aufwand, der quadratisch zur Benutzerzahl des WfMS wächst. Da dadurch die Belastung für die WF-Server und für das Kommunikationssystem sehr groß werden würde, verbietet sich dieses Verfahren für WfMS mit einer sehr großen Anzahl von Benutzern.

2.5.3 Arbeitslistenverwaltung in ADEPT

Um eine optimale Arbeitslistenverwaltung zu erhalten, wurden die Vorteile der beiden vorgestellten Verfahren kombiniert: Der Aufwand ist maximal proportional zur Anzahl der Benutzer des WfMS und eine Kommunikation mit einem Client findet nur dann statt, wenn sich die Arbeitsliste des zugehörigen Benutzers tatsächlich verändert hat. Um diese äußerst positiven Eigenschaften zu erreichen, wird die Übertragung einer veränderten Arbeitsliste an einen WF-Client eine gewisse Zeit verzögert und die Änderungen werden geblockt übertragen. Dadurch ergibt sich eine Last, die maximal proportional zur Anzahl der Benutzer des WfMS ist, da es für Übertragungen zum selben Benutzer einen Mindestzeitabstand gibt. Die Last ist also höchstens so groß wie bei einem periodischen Polling. Andererseits findet keine Übertragung statt, wenn sich die Arbeitsliste des Benutzers nicht verändert hat. Diese Optimierung ist in dem Sinne flexibel, dass für jeden Aktivitätentyp eine maximale Verzögerungszeit definiert werden kann, so dass Anforderungen an den Aktualitätsbedarf weiterhin erfüllt werden können.

Die soeben beschriebene Vorgehensweise führt zu einem sehr guten Kommunikationsverhalten und ermöglicht damit ein effizientes WF-Management. Deshalb wird dieses Verfahren für ADEPT gewählt und im Folgenden unterstellt.

2.6 Zusammenfassung

In diesem Kapitel wurde die Modellierung des Kontrollflusses in ADEPT beschrieben und es wurde die zugehörige Ausführungssemantik definiert. Beides ist so gestaltet, dass zur Ausführungszeit der WF-Instanzen möglichst viel Information über deren Zustand verfügbar ist. Deshalb werden zur Modellierung explizite Konstrukte z.B. für Verzweigungen und Schleifen verwendet und der Zustand der einzelnen Aktivitäten wird direkt durch den Kontrollfluss-Ausführungsgraphen angegeben. Wird noch weitergehende Information benötigt, wie z.B. der Bearbeiter eine Aktivität, so kann zusätzlich auf Ablaufhistorien zurückgegriffen werden. Beim Entwurf des Konzeptes für den Datenfluss von ADEPT$_{base}$ wurde darauf geachtet, dass eine effiziente verteilte WF-Ausführung ermöglicht wird, weshalb z.B. Abhängigkeiten zwischen den Daten parallel ausgeführter Zweige vermieden wurden. Ein weiterer sehr wichtiger Aspekt ist, dass die Konsistenz des Datenflusses schon durch die Modellierung garantiert werden kann, so dass Probleme zur Ausführungszeit vermieden werden. Diese wären wegen ihrer Häufigkeit in einem System mit sehr vielen Benutzern kaum zu bewältigen.

Außerdem wurden in diesem Kapitel die Bearbeiterzuordnungen der Aktivitäten eingeführt. Besonderes Augenmerk wurde dabei auf abhängige Bearbeiterzuordnungen gelegt, da sie in der Praxis sehr relevant sind und in der Literatur bisher kaum beachtet wurden. Insbesondere wird bei Ansätzen für verteiltes WF-Management häufig vereinfachenderweise angenommen, dass die OE der für eine Aktivitäteninstanz qualifizierten Bearbeiter schon zur Modellierungszeit feststeht (z.B. bei MENTOR [WWWK96b, MWW$^+$98]). Ein weiterer wichtiger Gesichtspunkt des WF-Modells ist das zur Aktualisierung der Arbeitslisten gewählte Verfahren. Dieses hat einen großen Einfluss auf die entstehenden

Kommunikationskosten. Deshalb wurde in diesem Kapitel ein entsprechendes Verfahren für ADEPT entwickelt, durch das sich die Anzahl der notwendigen Aktualisierungen drastisch senken lässt und bei dem trotzdem die Aktualität der Arbeitslisten garantiert werden kann.

Teil II

Verteilte Workflow-Ausführung in ADEPT

Kapitel 3

Partitionierung und Migration

Wir betrachten in dieser Arbeit WfMS, die wegen ihrer hohen Benutzerzahl eine große Last zu bewältigen haben. In einem solchen Szenario muss die Steuerung der WF-Instanzen sehr effizient erfolgen. Das heißt, die verteilte WF-Ausführung muss so realisiert werden, dass die Belastung des Kommunikationssystems möglichst gering ist. Um dies umzusetzen, wird eine geeignete Systemumgebung benötigt. Das Kommunikationssystem muss also so aufgebaut sein, dass eine effiziente WF-Steuerung ermöglicht wird. Außerdem müssen die Systemkomponenten (WF-Server und -Clients) auf eine intelligente Art und Weise im Kommunikationssystem angeordnet sein. Welche Anordnung dabei am besten geeignet ist, wird im vorliegenden Kapitel untersucht und es wird gezeigt, dass durch diese Anordnung die Belastung des Kommunikationssystems reduziert werden kann. Auf Basis des resultierenden – geeignet strukturierten – verteilten Systems, ist eine effiziente Verteilung der zu bewältigenden Aufgaben auf die WF-Server möglich. Es wird untersucht, wie eine solche Verteilung auszusehen hat, und es werden die zu ihrer Realisierung notwendigen Verfahren entwickelt. Dabei wird angenommen, dass unabhängige Bearbeiterzuordnungen (vgl. Abschnitt 2.4.2) deutlich dominieren, was zu der Verwendung von statischen Serverzuordnungen führt. Die in diesem Kapitel gemachten Aussagen gelten aber auch für den Fall abhängiger Bearbeiterzuordnungen (vgl. Abschnitt 2.4.3) und variabler Serverzuordnungen (siehe Kapitel 5), falls dies nicht explizit ausgeschlossen wird.

Im Abschnitt 3.1 wird eine Systeminfrastruktur entwickelt, die einen hohen Grad an Optimierungspotential bei der Festlegung einer konkreten Verteilung für eine WF-Instanz bietet. In Abschnitt 3.2 wird die grundsätzliche Strategie bei der Aufgabenverteilung festgelegt. Dazu wird das Konzept der WF-Partitionierung entwickelt. Die Partitionierung macht Migrationen erforderlich. Wie diese realisiert werden können und wie dies effizient geschehen kann, wird in Abschnitt 3.3 diskutiert. Außerdem wird untersucht, welchen Einfluss Migrationen auf das Zurücksetzen von WF-Instanzen haben. In Abschnitt 3.4 werden Alternativen zu der in diesem Kapitel vorgestellten Vorgehensweise diskutiert. Das Kapitel schließt mit einer Zusammenfassung in Abschnitt 3.5.

3.1 Physische Verteilung der Systemkomponenten

Im Folgenden wird die dem WfMS zugrunde liegende Systeminfrastruktur betrachtet, d.h. der Aufbau des dem WfMS zugrunde liegenden Kommunikationssystems und die Anordnung der Komponenten des WfMS innerhalb dieses Kommunikationssystems. Es wird untersucht, wie diese beschaffen sein

muss, damit das Ziel, die Kommunikationslast zu reduzieren, erreicht werden kann. Für die Kommunikationslast ist die durchschnittliche Last pro Teilnetz relevant, und nicht die maximal pro Teilnetz oder insgesamt kommunizierte Datenmenge. Jedoch ist darauf zu achten, dass keine Flaschenhälse im Kommunikationssystem entstehen, da diese zur Überlastung der entsprechenden Teilnetze führen können.

Um das Problem besser zu verstehen, wird zuerst analysiert, durch welche Aktionen ein WF-Server bzw. das Kommunikationssystem belastet wird. Dann wird, ausgehend von einer zentralen Architektur, eine Systeminfrastruktur entwickelt, welche die gestellten Anforderungen erfüllt. Schließlich wird noch gezeigt, dass sich durch diese Infrastruktur die durchschnittliche Belastung der Teilnetze tatsächlich reduzieren lässt, und dass Flaschenhälse vermieden werden.

3.1.1 Aufgaben des WF-Servers

Um zu erläutern, warum ein zentraler WF-Server schon bei einer relativ geringen Anzahl von Benutzern und WF-Instanzen überlastet ist, untersuchen wir nun, welche Aufgaben er zu bewältigen hat. Der WF-Server muss für jede auszuführende Aktivitäteninstanz u.a. die folgenden Aufgaben erledigen:

- Für jeden Benutzer des WfMS muss einzeln überprüft werden, ob dieser die Bearbeiterzuordnung der anstehenden Aktivität erfüllt. Eine Vorberechnung der für eine bestimmte Aktivität geeigneten Bearbeiter ist i.Allg. nicht möglich, da diese von den Bearbeitern vorangehender Aktivitäten abhängen können (bei abhängigen Bearbeiterzuordnungen). Außerdem verändert sich die Menge der potentiellen Bearbeiter einer Aktivität, wenn sich die Menge der Benutzer des WfMS ändert.
- Vor dem Start jedes Aktivitätenprogramms müssen dessen Eingabeparameterdaten aus der Datenbank des WfMS geladen werden. Nach Beendigung des Programms müssen die Ausgabeparameterdaten zurückgeschrieben werden. Je nach Anwendung kann es sich dabei um recht große Datenmengen handeln.

Ein WF-Server ist in zahlreiche und umfangreiche Kommunikationen involviert. Dies sind für jede auszuführende Aktivitäteninstanz mindestens die in Abb. 3.1 dargestellten Aktionen:

- Die Information, dass die Aktivitäteninstanz zur Ausführung ansteht, muss zu den Rechnern aller potentiellen Bearbeiter (WF-Clients) transportiert werden, damit die Aktivitäteninstanz in deren Arbeitsliste aufgenommen werden kann.
- Wenn sich ein Benutzer entschlossen hat, die Aktivitäteninstanz zu bearbeiten, so wird dies dem WF-Server mitgeteilt. Falls solche Anforderungen von mehreren Benutzern eingehen, so werden diese vom WF-Server synchronisiert. Nur die erste eingehende Anforderung wird bestätigt, alle weiteren werden zurückgewiesen.
- Da der Bearbeiter der Aktivitäteninstanz nun feststeht, muss der entsprechende Eintrag wieder aus den Arbeitslisten entfernt werden. Dies erfordert eine Kommunikation zu allen potentiellen Bearbeitern der Aktivitäteninstanz.
- Bevor das Aktivitätenprogramm gestartet werden kann, müssen die Eingabeparameter zu demjenigen Rechner transportiert werden, auf dem das Programm ablaufen soll.
- Nach Beendigung des Aktivitätenprogramms werden die Ausgabeparameter zum WF-Server zurücktransportiert.

Bei Kommunikationen, die Arbeitslisteneinträge betreffen, werden i.d.R. keine besonders großen Datenmengen transferiert. Da für jede auszuführende Aktivitäteninstanz ein Eintrag in die Arbeitsliste

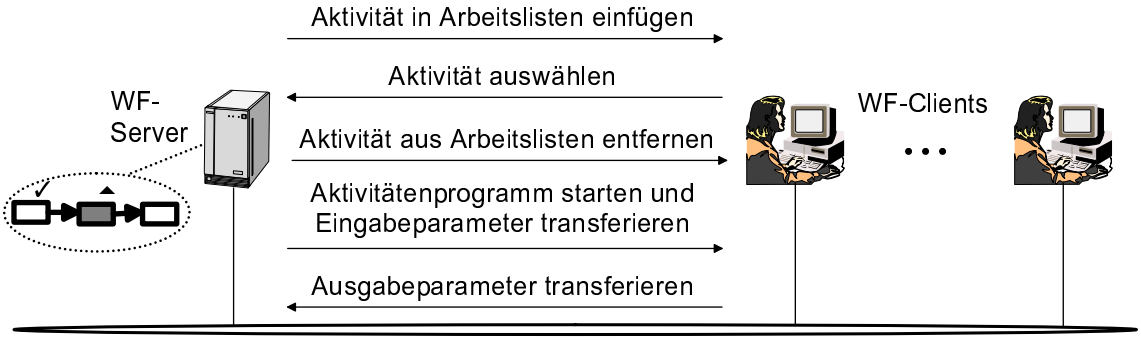


Abbildung 3.1 Kommunikationen, in die der WF-Server involviert ist.

jedes potentiellen Bearbeiters eingefügt und wieder herausgenommen werden muss, entstehen durch solche Aktionen jedoch sehr viele Nachrichten. Die Parameterdaten einer Aktivität können sehr umfangreich sein (siehe das Zahlenbeispiel aus Abschnitt 1.2.2). Deshalb werden beim Transfer von Parameterdaten große Datenmengen bewegt. Beide Aspekte führen zusammen dazu, dass das Kommunikationssystem sowohl mit einer großen Datenmenge, als auch mit einer hohen Anzahl von Nachrichten belastet wird. Aus diesem Grund ist es bei WfMS besonders wichtig, dass eine geeignete Systeminfrastruktur verwendet wird, um die Kommunikationslast bewältigen zu können. Dies bestätigt die in dem Zahlenbeispiel aus Abschnitt 1.2.2 und in [BD97, BD98, BD00a] gewonnene Erkenntnis, dass ein LAN schon bei moderaten Benutzerzahlen überlastet sein kann. Unter Berücksichtigung dieses Aspekts wird in den folgenden Abschnitten eine geeignete Infrastruktur entwickelt.

3.1.2 Zentraler WF-Server

Wie in Abb. 3.2 dargestellt, sind die Benutzer des WfMS (Clients) in einem Kommunikationssystem verteilt. Dieses Netzwerk sollte vollständig vermascht sein, um „Durchgangsverkehr“ in den Teilnetzen zu vermeiden. Der WF-Server legt seine Daten in einer WF-DB ab. Wenn sich diese auf demselben Rechner befinden, wie der WF-Server, belastet die Kommunikation zwischen den beiden das Netzwerk nicht. Allerdings wird der Rechner des WF-Servers dann zusätzlich durch die WF-DB belastet. Deshalb kann es sinnvoll sein, die WF-DB auf einem anderen Rechner zu platzieren, der sich aber im selben Teilnetz wie der WF-Server befindet. Diese Variante sollte immer dann gewählt werden, wenn der Rechner des WF-Servers an seine Leistungsgrenze stößt, das entsprechende Teilnetz aber noch nicht ausgelastet ist.

In den Teilnetzen können sich auch externe Datenquellen (siehe Abschnitt 2.3.1) befinden. Dies sind z.B. Datenbanken, die Anwendungsdaten verwalten, welche von den Aktivitätenprogrammen verwendet werden. Diese Daten fließen direkt zwischen Datenquelle und Aktivitätenprogramm. Deshalb sind Aspekte, wie die Platzierung eines WF-Servers, für die dadurch entstehenden Kommunikationskosten nicht relevant. Bei der Beurteilung einer Systeminfrastruktur werden externe Datenquellen deshalb nicht einbezogen.

Betrachten wir nun den Fall eines zentralen WF-Servers. Das heißt, es gibt nur einen einzigen WF-Server, der sich in genau einem Teilnetz befindet. Da, wie im vorherigen Abschnitt erläutert wurde, für den WF-Server eine recht große Last entsteht, stößt ein zentraler Server schon bei einer relativ geringen Benutzerzahl an seine Leistungsgrenzen. Die Beseitigung bzw. Abmilderung dieses potentiellen Engpasses erfordert relativ rasch den Einsatz von (teuren) Hochleistungssystemen, z.B. in Form von Mehrprozessorsystemen und von Mehrrechner-DBMS. Ob sich damit die erforderliche Leistung

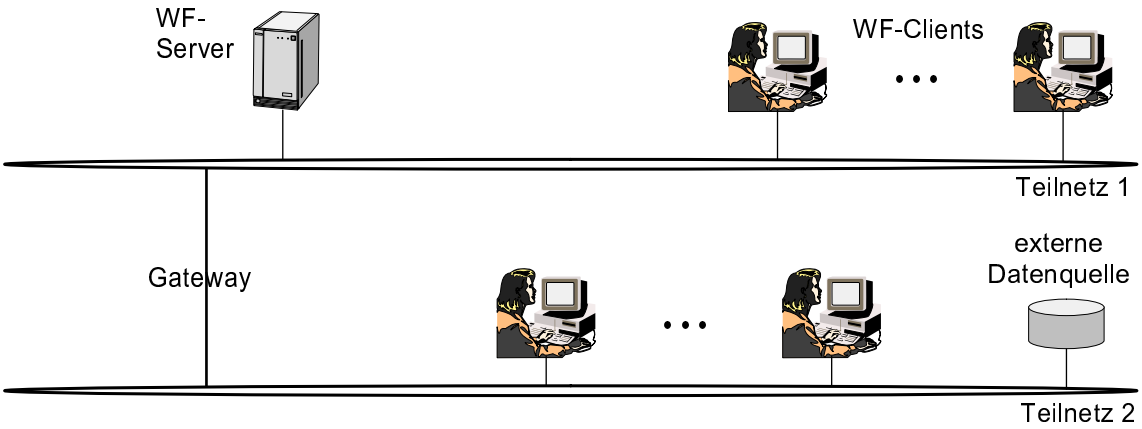


Abbildung 3.2 Kommunikationssystem mit einem zentralen WF-Server.

erzielen lässt, hängt stark davon ab, inwieweit man die Steuerung der einzelnen WF-Instanzen sowie deren Daten so auf die beteiligten Teilsysteme verteilen kann, dass eine gute Lastbalancierung erreicht wird und wenig Zugriffskonflikte auftreten. Dieser Ansatz führt aber zu sehr hohen Kosten. Außerdem entstehen trotzdem die im nachfolgenden Abschnitt beschriebenen Probleme. Deshalb ist es vorteilhafter mehrere „kleine“ Rechner zu verwenden und die Last auf eine intelligente Art und Weise auf sie zu verteilten.

3.1.3 Replizierte WF-Server in einem Teilnetz

Um das Problem eines überlasteten WF-Servers zu lösen, ist die naheliegendste Lösung, ihn einfach zu replizieren (siehe Abb. 3.3). Da die Gesamtlast jetzt von mehreren WF-Servern gemeinsam bewältigt wird, löst dies (im Prinzip) das Problem der Serverüberlastung. Jeder dieser WF-Server verwendet eine separate WF-DB, um Flaschenhälse beim DB-Zugriff zu vermeiden. Da es nun mehrere Server im WfMS gibt, stellt sich die Frage, mit welchem/n ein bestimmter WF-Client eine Verbindung aufbauen soll, so dass er von diesem Server kontrollierte Aktivitäten bearbeiten kann. Die nahe liegendste Idee, dass sich ein Client nur beim Server seines eigenen Teilnetzes anmeldet, ist zu restriktiv. Dies würde erzwingen, dass sich alle potentiellen Bearbeiter einer Aktivität im selben Teilnetz befinden müssen, was in der Realität aber nicht immer gegeben ist. Deshalb muss sich ein Client prinzipiell bei allen WF-Servern anmelden. Falls es jedoch Server gibt, die für ihn nicht relevant sind, weil sie niemals Aufgaben für diesen Client kontrollieren, so wird auf die entsprechende Verbindung verzichtet. Ein Beispiel hierfür ist ein Angestellter der Fertigung und ein Server der Finanzbuchhaltung, wenn diese beiden OE z.B. ausschließlich in disjunkte WF-Typen involviert sind. Da Benutzer i.d.R. äußerst selten von Servern fremder OE kontrollierte Aufgaben bearbeiten, ist meist eine Verbindung mit recht wenigen WF-Servern ausreichend. Welche dies sind, ist durch eine entsprechende Konfiguration für den WF-Client vorzugeben.

Bei dieser Art der Replikation befinden sich alle WF-Server, wie in Abb. 3.3 dargestellt, im selben Teilnetz (z.B. dem des Rechenzentrums). Da in alle in einem WfMS ablaufenden Kommunikationen ein WF-Server involviert ist und sich alle WF-Server im Teilnetz 1 befinden, wird dieses Teilnetz mit allen Kommunikationen des WfMS belastet und bildet deshalb einen Flaschenhals. In Abschnitt 1.2.2 wurde schon gezeigt, dass ein LAN schon bei moderaten Benutzerzahlen eines WfMS überlastet sein kann. Deshalb ist diese Systeminfrastruktur wenig geeignet. Hinzu kommt, dass weit(er) entfernte Benutzer in der Regel mit schlechteren Antwortzeiten bedient werden.

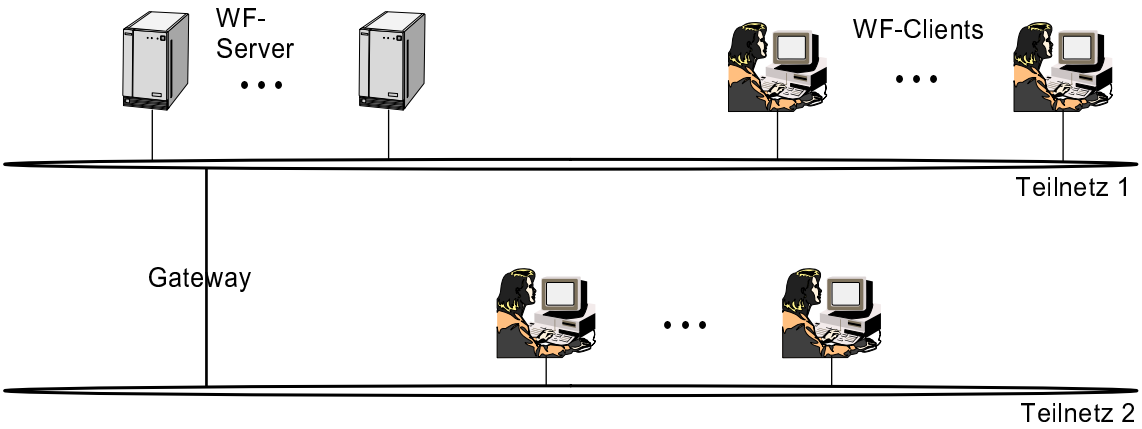


Abbildung 3.3 Replizierte WF-Server in einem Teilnetz.

3.1.4 WF-Server auf Teilnetze verteilt

Um den soeben beschriebenen Kommunikationsengpass zu vermeiden, sollte sich in jedem Teilnetz ein WF-Server befinden (vgl. Abb. 3.4). Dieser kann allenfalls in Einzelfällen entfallen, falls festgestellt wird, dass er nicht benötigt wird. Durch diese Systeminfrastruktur lässt sich die Last nicht nur auf die WF-Server sondern auch auf die Teilnetze verteilen. Da alle Teilnetze gleichartig sind, gibt es in einem solchen System auch keine Flaschenhälse. Im nächsten Abschnitt wird diese Eigenschaft noch an einigen Beispielen demonstriert. Außerdem wird erläutert, warum durch diese Systeminfrastruktur auch die durchschnittliche Belastung der Teilnetze reduziert wird. Aufgrund dieser Vorteile wird diese Systeminfrastruktur in der weiteren Arbeit verwendet. In Definition 3.1 wird dazu noch der Begriff des Domains eingeführt.

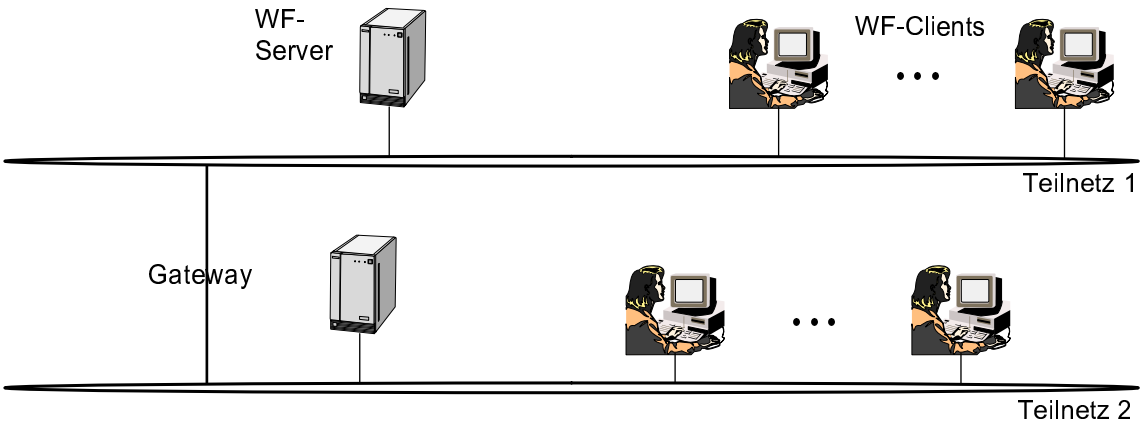


Abbildung 3.4 Ein WF-Server in jedem Teilnetz.

Definition 3.1 (Domain)
Ein Teilnetz, zusammen mit den darin enthaltenen WF-Servern[1], Clients und externen Datenquellen wird als *Domain* bezeichnet.

[1]Bis auf weiteres wird angenommen, dass sich in jedem Teilnetz nur ein WF-Server befindet. Ausnahmen von dieser Regel werden in Kapitel 7 diskutiert.

Obwohl alle Teilnetze gleichartig sind (vgl. Abb. 3.4), unterscheiden sich die WF-Server durch die konkret zu ihrem Domain gehörenden Benutzer. Da der Großteil der Kommunikation eines WF-Servers mit den Benutzern (Clients) stattfindet, ist es für die Kommunikationslast entscheidend, welcher WF-Server eine bestimmte Aufgabe übernimmt. In Abschnitt 3.2 wird deshalb eine intelligente Strategie zur Aufgabenverteilung entwickelt.

3.1.5 Analyse der gewählten Systeminfrastruktur

Ziel dieses Abschnitts ist es, zu zeigen, dass die gewählte Kommunikationsinfrastruktur (siehe Abschnitt 3.1.4) dazu beiträgt, die durchschnittliche Belastung der Teilnetze zu reduzieren. Eine ähnliche Betrachtung findet sich in [BD97]. Außerdem soll gezeigt werden, dass sich im Kommunikationssystem keine Flaschenhälse befinden.

Die Analyse dient dazu, die Güte der gewählten Kommunikationsinfrastruktur zu zeigen. Es ist nicht das Ziel, konkrete Aufgabenverteilungen zu untersuchen. Deshalb wird vereinfachend angenommen, dass alle WF-Server im Beobachtungszeitraum dieselbe Anzahl von Aktivitäteninstanzen kontrollieren. Außerdem sollen alle diese Aktivitäteninstanzen dasselbe Datenvolumen benötigen und von gleich vielen Benutzern bearbeitet werden können. Wird diese Gleichverteilung der Last angenommen, so besteht keine Notwendigkeit, Nachrichten oder Datenpakete zu zählen. Stattdessen genügt es, in jedem Teilnetz die Anzahl der Verbindungen zwischen WF-Servern und Clients zu zählen, um ein Maß für die Belastung des Teilnetzes zu erhalten. So ist das Teilnetz 1 in Abb. 3.5a mit 6 Verbindungen belastet und in Abb. 3.5b mit nur 2 Verbindungen, was 1/3 der Last bedeutet.

Im Folgenden wird der Fall analysiert, dass 3 Prozesse von jeweils 2 Bearbeitern ausgeführt werden. Die Kommunikationen der verschiedenen Prozesse werden in den Abbildungen durch unterschiedliche Linienarten für die Verbindungen dargestellt. In Abb. 3.5a - 3.7a werden die Prozesse alle vom selben zentralen WF-Server kontrolliert, wohingegen in Abb. 3.5b - 3.7b jeder Prozess von einem anderen Server kontrolliert wird. Für den Fall mit mehreren WF-Servern werden die folgenden 3 Szenarien unterschieden:
Szenario 1: Alle Benutzer befinden sich im selben Teilnetz wie der zugehörige WF-Server.
Szenario 2: Die Hälfte der Benutzer befinden sich im selben Teilnetz wie der WF-Server, die andere Hälfte befindet sich in einem „falschen Teilnetz“.
Szenario 3: Alle Benutzer befinden sich im „falschen Teilnetz“.
Wir untersuchen für diese Szenarien die maximale Belastung eines Teilnetzes (um Flaschenhälse zu erkennen) und die durchschnittliche Belastung der Teilnetze. Dabei werden die Werte des zentralen Falls mit denen des verteilten Falls verglichen.

Szenario 1: Wie in Abb. 3.5b dargestellt, befinden sich in diesem Szenario die Clients bei Verwendung mehrerer WF-Server stets im selben Teilnetz wie der zugehörige WF-Server. Die Gateways werden nicht benutzt und jedes Teilnetz wird mit 2 Verbindungen belastet. Wird ein zentraler WF-Server verwendet (Abb. 3.5a), so belasten alle 6 Verbindungen das Teilnetz 1 des WF-Servers. Die anderen beiden Teilnetze werden mit 2 Verbindungen belastet. Insgesamt laufen 4 Verbindungen über ein Gateway. In diesem Idealfall wird also durch die Verwendung mehrerer WF-Server die maximale Belastung eines Teilnetzes von 6 auf 2 Verbindungen reduziert. Die durchschnittliche Belastung sinkt von 3,33 auf 2 Verbindungen. Wenn n Teilnetze mit n WF-Servern verwendet werden, dann sinkt bei einer derart optimalen Verteilung der Benutzer die maximale Belastung eines Teilnetzes auf $1/n$-tel der Last des zentralen Falls. Der Grund dafür ist, dass alle Kommunikationen nur ein Teilnetz belasten, weshalb sich die Verbindungen auf die n Teilnetze aufteilen.

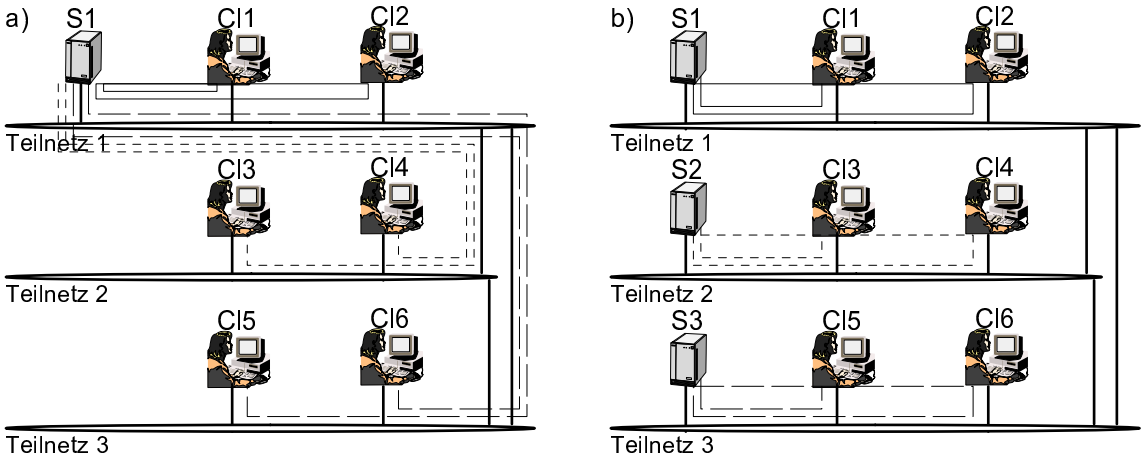


Abbildung 3.5 Alle Benutzer befinden sich im „richtigen Teilnetz“.

Szenario 2: Befindet sich die Hälfte der Bearbeiter im „richtigen Teilnetz“ (Abb. 3.6), so wird im verteilten Fall jedes Teilnetz mit 3 Verbindungen belastet. Die Gateways werden mit insgesamt 3 Verbindungen belastet. Im zentralen Fall ergibt sich dieselbe Last wie im Szenario 1. Das bedeutet also, dass selbst wenn sich nicht die Mehrheit der Benutzer im richtigen Teilnetz befindet, die maximale Belastung eines Teilnetzes von 6 auf 3 Verbindungen halbiert wird und die durchschnittliche Belastung von 3,33 auf 3 Verbindungen sinkt.

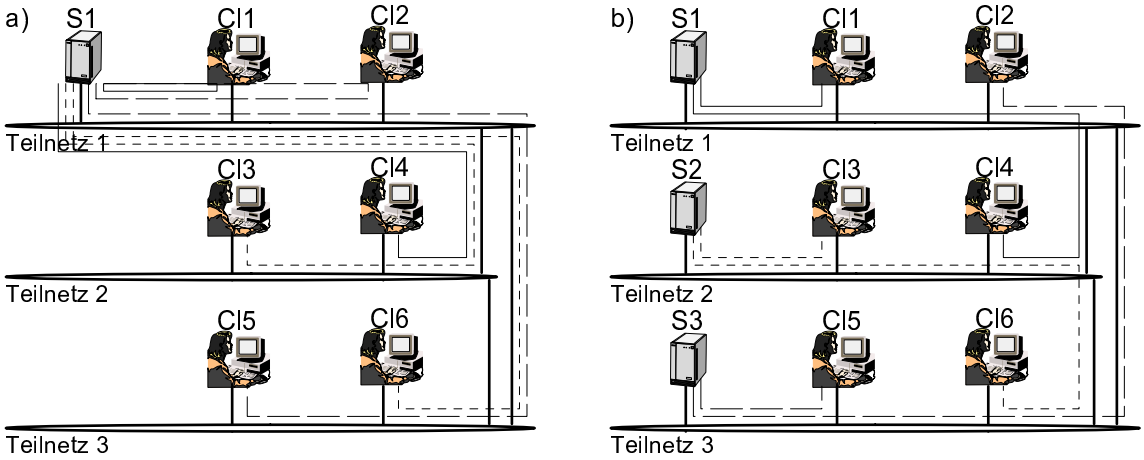


Abbildung 3.6 Die Hälfte der Benutzer befinden sich im „richtigen Teilnetz“.

Szenario 3: Dieses äußerst ungünstige Szenario, in dem sich kein einziger Bearbeiter im „richtigen Teilnetz“ befindet, kann stets durch eine andere Verteilung der Aufgaben auf die WF-Server vermieden werden. Es soll dennoch analysiert werden, um die Vorteile der gewählten Systeminfrastruktur zu verdeutlichen. Im verteilten Fall wird jedes Teilnetz mit 4 Verbindungen belastet. Damit sinkt die maximale Belastung eines Teilnetzes von im zentralen Fall 6 auf 4 Verbindungen. Die Bildung eines Flaschenhalses wird also sogar in diesem äußerst ungünstigen Szenario verhindert. Die durchschnittliche Belastung der Teilnetz steigt allerdings von 3,33 auf 4 Verbindungen. Auch die Gesamtlast der Gateways steigt von 4 auf 6 Verbindungen. Dies liegt daran, dass sich im zentralen Fall immerhin 2 Benutzer im selben Teilnetz wie der WF-Server befinden.

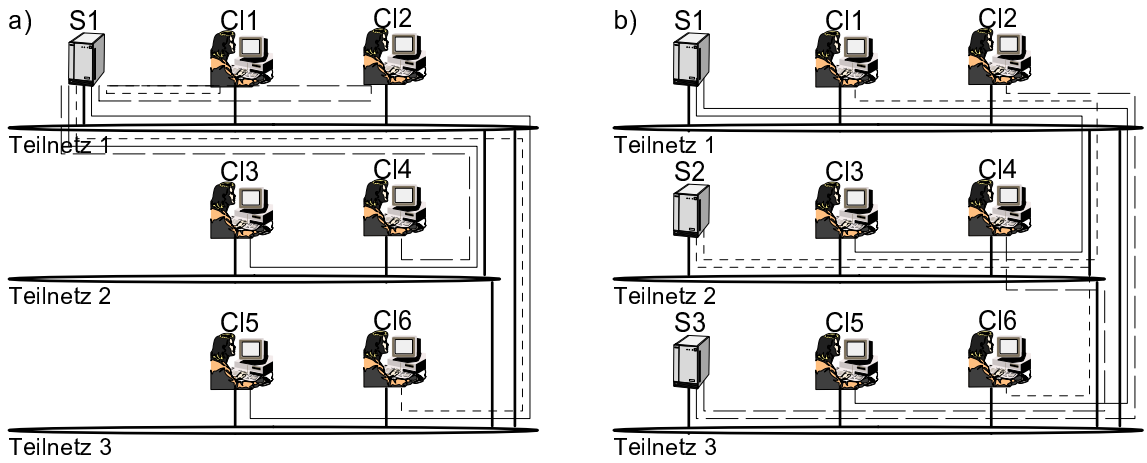


Abbildung 3.7 Alle Benutzer befinden sich im einem „falschen Teilnetz“.

Zusammenfassend lässt sich feststellen, dass durch die gewählte Kommunikationsinfrastruktur Flaschenhälse stets vermieden werden. Dies gilt sogar dann, wenn sich alle Bearbeiter der Aktivitäten in einem „falschen Teilnetz“ befinden. Die Ergebnisse bzgl. der maximalen und der durchschnittlichen Belastung der Teilnetze sind umso besser, je mehr Benutzer sich lokal im Teilnetz des für sie relevanten WF-Servers befinden. Dasselbe gilt für die Belastung der Gateways. Deshalb muss es das Ziel sein, eine Aufgabe jeweils demjenigen WF-Server zuzuordnen, der eine möglichst große Lokalität zu den Bearbeitern der Aufgabe aufweist.

3.2 Verteilung der Aufgaben auf die Workflow-Server

Die verschiedenen für die Bearbeitung einer Aufgabe zur Verfügung stehenden WF-Server unterscheiden sich durch das Teilnetz, in welchem sie sich befinden. Dadurch haben sie zu den verschiedenen Benutzern des WfMS eine unterschiedlich hohe Lokalität. Wie wir im vorherigen Abschnitt gesehen haben, hat die Wahl des WF-Servers für eine bestimmte Aufgabe deshalb Auswirkungen auf die bei ihrer Ausführung entstehenden Kommunikationskosten. Daraus folgt, dass es extrem wichtig ist, die anstehenden Aufgaben geeignet auf die WF-Server zu verteilen. Deshalb wird in den folgenden Abschnitten ein geeignetes Verfahren zur Verteilung der Aufgaben entwickelt. Dieses wird im Folgenden als das Verteilungsmodell von ADEPT bezeichnet.

3.2.1 Ziel bei der Verteilung der Aufgaben

Bevor auf Details der Aufgabenverteilung eingegangen wird, muss erst das generelle Ziel geklärt werden. Dies erfolgt im vorliegenden Abschnitt. In den darauf folgenden Abschnitten wird dann untersucht, wie dieses Ziel erreicht werden kann.

Das oberste Ziel bei der Aufgabenverteilung muss sein, für jede Aufgabe einen WF-Server zu verwenden, der eine hohe Lokalität zu den zugehörigen Bearbeitern aufweist. Am günstigsten ist natürlich derjenige WF-Server, der sich im selben Domain wie diese Bearbeiter befindet. Diese Strategie hat zahlreiche Vorteile:

- Die Kommunikationskosten werden reduziert: Wie schon in Abschnitt 3.1.5 gezeigt wurde, wird durch eine hohe Lokalität zwischen dem WF-Server und den Bearbeitern einer Aktivität die durchschnittliche Belastung der Teilnetze reduziert. Auch die Belastung der Gateways ist geringer, wenn sich ein Großteil der Bearbeiter im selben Teilnetz wie der WF-Server befindet. Ist es nicht möglich, denjenigen WF-Server zu verwenden, der sich im Domain der Bearbeiter befindet, so sollte wenigstens ein WF-Server verwendet werden, der sich „nahe“ bei diesen Bearbeitern befindet. Dadurch wird für die Kommunikation zwischen WF-Server und Clients kein WAN-Gateway benötigt. Stattdessen müssen die Daten nur über eine kurze Distanz transportiert werden, was zur Reduzierung der Kommunikationskosten beiträgt.
- Die Antwortzeiten und damit die Verzögerungen für die Benutzer werden reduziert: Da eine weit entfernte Kommunikation i.d.R. länger dauert als eine Kommunikation innerhalb desselben LAN oder zu einem benachbarten LAN, reduziert die gewählte Strategie die Übertragungszeiten. Deshalb müssen die Benutzer keine so langen Wartezeiten in Kauf nehmen, wenn sie z.B. einen Eintrag aus ihrer Arbeitsliste ausgewählt haben und auf die Übertragung der Eingabeparameter warten.
- Die Verfügbarkeit des WfMS wird erhöht: Da Gateways innerhalb eines Standortes i.d.R. eine höhere Verfügbarkeit aufweisen als WAN-Verbindungen, ist ein lokaler WF-Server für die Benutzer häufiger erreichbar, als ein weit entfernter. Dies gibt insbesondere dann, wenn sich der WF-Server, der die Ausführung einer Aufgabe steuert, im selben Domain befindet wie der Benutzer. Dann ist nur die Verfügbarkeit des entsprechenden LAN und des WF-Servers Voraussetzung für die Ausführbarkeit der Aufgabe. Es wird dazu kein Gateway benötigt.
- Eine hohe Lokalität zwischen dem WF-Server und den Bearbeitern der Aufgaben wird bei unternehmensübergreifenden WfMS häufig zwingend verlangt: Da bei Kooperationen [Pic98] kein Unternehmen auf den WF-Server eines anderen Unternehmens angewiesen sein will, muss jede Aufgabe von einem WF-Server kontrolliert werden, der sich zumindest im selben Unternehmen befindet, wie die entsprechenden Bearbeiter.

3.2.2 Verteilung von Workflows als Ganzes

Die naheliegendste Möglichkeit, um Aufgaben auf mehrere WF-Server zu verteilen, ist, einen kompletten WF stets einem WF-Servern zuzuordnen. Im Folgenden wird diskutiert, welche Vorgehensweisen dafür möglich sind. Anschließend werden dabei entstehende Probleme analysiert.

Prinzipiell gibt es die beiden Möglichkeiten, die Prozesse auf Typ- oder auf Instanzebene den WF-Servern zuzuordnen. Bei einer Zuordnung auf Typebene wird für jeden WF-Typ zur Modellierungszeit der günstigste WF-Server bestimmt. Dies ist derjenige, der sich am nächsten bei den potentiellen Bearbeitern dieses WF befindet. Dieser WF-Server steuert dann alle WF-Instanzen dieses Typs. Ein ähnliches Verfahren wird z.B. in MOBILE [HS96] verwendet. Bei einer Verteilung auf Instanzebene wird für jede WF-Instanz zur Ausführungszeit entschieden, welcher WF-Server sie kontrolliert. Eine einfache Möglichkeit hierfür wäre, denjenigen WF-Server auszuwählen, der sich im selben Domain befindet, wie der Benutzer, der die WF-Instanz gestartet hat. Eine Verteilung auf Instanzebene wird von Exotica/Cluster [AKA+94] praktiziert, wobei bei diesem Ansatz der WF-Server für eine WF-Instanz zufällig ausgewählt wird.

Die Verteilung von gesamten WF auf die WF-Server führt aber zu einem Problem, sobald es WF gibt, die mehrere OE durchlaufen. Dies soll an dem in Abb. 3.8 dargestellten Verkaufs-WF erläutert werden. Die ersten 10 Aktivitäten betreffen Verkaufsverhandlungen mit einem Kunden. Sie werden

„vor Ort“ beim Kunden in Brasilien ausgeführt. Anschließend erfolgt der Versand zentral für ganz Amerika aus den USA. Schließlich werden die Abrechnung und das Verschicken der Rechnungen in der Konzernzentrale in Deutschland durchgeführt.

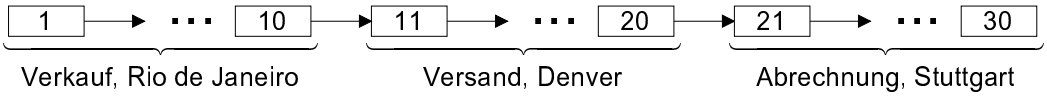


Abbildung 3.8 Beispiel für einen WF, der nacheinander mehrere OE durchläuft.

Da der WF von nur einem Server kontrolliert wird, kann sich dieser entweder in Rio de Janeiro, Denver oder Stuttgart befinden. Das bedeutet, egal wie intelligent die Strategie zur Auswahl des WF-Servers ist, für 2/3 der Aktivitäten ist der Server sehr weit von den Bearbeitern entfernt. Für diese Aktivitäten entstehen hohe Kommunikationskosten und lange Wartezeiten für die Benutzer. Da dies normalerweise nicht akzeptabel ist, wird im nun folgenden Abschnitt ein Ansatz vorgestellt, der diese Nachteile vermeidet.

3.2.3 Partitionierung der Workflows

Da soeben gezeigt wurde, dass die Zuordnung gesamter WF zu den WF-Servern ein ungeeignetes Verfahren ist, wird nun ein Modell entwickelt, das es erlaubt, verschiedene Abschnitte einer WF-Instanz von unterschiedlichen Servern kontrollieren zu lassen.

Um das im vorherigen Abschnitt geschilderte Problem zu lösen, muss ein WF logisch in Abschnitte (sogenannte Partitionen) zerlegt werden, die von unterschiedlichen WF-Servern kontrolliert werden. Ein solches Vorgehen wird z.B. auch in [CGS97, DKM+97, MWW+98] beschrieben. In ADEPT$_{distribution}$ soll jede Partition von einem WF-Server kontrolliert werden, der sich nahe bei den potentiellen Bearbeitern der Aktivitäten der Partition befindet. Dieser WF-Server wird durch eine sogenannte (statische) Serverzuordnung festgelegt. Für jede Aktivität n wird durch $ServZuordn_n$ angegeben, welcher Server diese Aktivität kontrollieren soll. Eine Partition ist damit ein zusammenhängender Teilgraph des Kontrollflussgraphen, der aus Aktivitäten besteht, bei denen aufgrund der Serverzuordnungen derselbe Server gewählt wird. Die Serverzuordnungen der Aktivitäten werden zur Modellierungszeit auf Typebene geeignet festgelegt. Die zu einer Aktivität gehörende Serverzuordnung wird als Attribut des Knotens gespeichert und zusammen mit der WF-Vorlage auf jedem Server des WfMS repliziert. Der WF wird nicht physisch zerteilt. Wie geeignete Serverzuordnungen ermittelt werden können, wird ausführlich in Kapitel 4 diskutiert.

Abb. 3.9 zeigt ein Beispiel für einen partitionierten WF. Die Partition 3 wird vom WF-Server 3 kontrolliert. So lautet die Serverzuordnung z.B. für die in ihr enthaltene Aktivität d: $ServZuordn_d =$ `Server 3`. Der Grund, weshalb diese Serverzuordnung so gewählt wird, ist, dass die Aktivität d hauptsächlich von Bearbeitern des Domain 3 ausgeführt wird. Die Serverzuordnungen werden in den in dieser Arbeit verwendeten Abbildungen jeweils unterhalb der zugehörigen Aktivität angegeben (Bearbeiterzuordnungen oberhalb).

Die Serverzuordnungen sind Ausdrücke, die für die Aktivitäten angeben, von welchem Server sie kontrolliert werden. Diese Ausdrücke sind momentan IDs von WF-Servern. Dies wird in Kapitel 5 erweitert, indem auch komplexere Ausdrücke als (variable) Serverzuordnungen zugelassen werden. $ServZuordn$ ist damit eine Funktion, welche die Menge der Knoten N des WF auf die Menge aller Ausdrücke *Ausdrücke* abbildet ($ServZuordn : N \mapsto$ *Ausdrücke*).

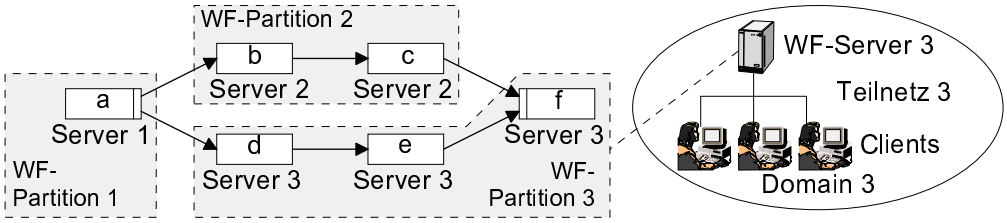


Abbildung 3.9 Beispiel für die Partitionierung eines WF.

Definition 3.2 (Server, der eine Aktivitäteninstanz kontrolliert)
S sei die Menge aller IDs der WF-Server des WfMS. Dann definiert die Funktion $Server$ für eine WF-Instanz aus $Inst$ und eine Aktivitäteninstanz aus N den WF-Server, der diese Aktivität kontrolliert: $Server : N \times Inst \mapsto S$ mit
$Server_{n,inst} = val_{inst}(ServZuordn_n)$
Das heißt, $Server_{n,inst}$ ist derjenige WF-Server, der sich zur Ausführungszeit durch die Auswertung von $ServZuordn_n$ ergibt. Falls aus dem Zusammenhang ersichtlich ist, welche WF-Instanz gemeint ist, so verwenden wir im Folgenden der Einfachheit halber $Server_n$.

$Server_n$ kann z.B. verwendet werden, um zur Ausführungszeit zu überprüfen, ob 2 Aktivitäteninstanzen vom selben Server kontrolliert werden: $Server_n = Server_m$. Dabei wird verglichen, ob die IDs der beiden ermittelten Server identisch sind. Es wird nicht geprüft, ob die Serverzuordnungsausdrücke $ServZuordn_n$ und $ServZuordn_m$ gleich sind (der Unterschied wird im Zusammenhang mit variablen Serverzuordnungen noch klarer). Ist gibt aber auch Fälle, in denen schon zur Modellierungszeit getestet werden muss, ob 2 Aktivitäten immer vom selben Server kontrolliert werden. Zu diesem Zweck dient die nachfolgend definierte Relation.

Definition 3.3 (Äquivalenz von Serverzuordnungen)
Die Serverzuordnungen der Aktivitäten n und m heißen äquivalent ($ServZuordn_n \equiv ServZuordn_m$), falls zur Ausführungszeit für alle WF-Instanzen $Server_n = Server_m$ gilt.[2]

3.3 Migration von Workflow-Instanzen

Eine partitionierte WF-Instanz wird von verschiedenen WF-Servern kontrolliert. Zwischen diesen wird Information über die WF-Instanz ausgetauscht. Außerdem migriert die Kontrolle über die WF-Instanz zwischen den Servern, d.h., wenn zur Ausführungszeit das Ende einer Partition erreicht wird, dann muss die Kontrolle an den Server der nächsten Partition übergeben werden. Damit dieser die Kontrolle übernehmen kann, müssen ihm der Zustand der WF-Instanz und die Werte der Datenelemente bekannt gemacht werden. In diesem Abschnitt wird geklärt, zu welchen Zeitpunkten Kommunikationen notwendig sind und wie diese (effizient) realisiert werden können. Allerdings wird in diesem Abschnitt nur das Grundprinzip der Migration entwickelt, optimierte Verfahren werden erst in Kapitel 6 vorgestellt.

[2] Für die in diesem Kapitel beschriebenen statischen Serverzuordnungen gilt dies genau dann, wenn die Serverzuordnungsausdrücke identisch sind. Der eigentliche Sinn dieser Definitionen zeigt sich erst in Kapitel 5, wenn als (variable) Serverzuordnungen auch komplexe Ausdrücke verwendet werden. Variable Serverzuordnungen können äquivalent sein, obwohl die Serverzuordnungsausdrücke verschieden sind.

3.3.1 Transaktionsschutz für Migrationen

Bei einer Migration werden Zustandsinformation der WF-Instanz und die Werte der Datenelemente übertragen. Dabei wäre es sehr problematisch, wenn eine Übertragung abgebrochen und die Ausführung mit den unvollständigen Daten fortgesetzt werden würde. Dann könnte es passieren, dass veraltete Datenelemente verwendet werden, weil die neuen noch nicht übertragen wurden. Außerdem könnte eine Wiederholung der Migration irrtümlich als die nächste Schleifeniteration interpretiert werden. Deshalb wird die Kommunikation bei einer Migration durch ein 2-Phasen-Commit-Protokoll (2PC) [MLO86, SBCM95] geschützt. Die Änderung des Zustandes der WF-Instanz, durch welche die Beendigung der Migration protokolliert wird, wird innerhalb derselben Transaktion wie die Datenübertragung durchgeführt. Dies lässt sich z.B. realisieren, indem innerhalb der DB-Transaktion ein Transactional Remote Procedure Call (TRPC) [GR93] abgesetzt wird. Ermöglicht wird dies durch die Verwendung von Transaktionsmonitoren [JH92, Lin95, Klö97] wie Encina [She93, IBM94] oder Tuxedo [ACM94, UNI92]. Eine Alternative hierzu wäre die Verwendung von Persistent Queues [GR93, MD94].

In beiden Fällen kann am Zustand der WF-Instanz erkannt werden, ob ein TRPC bzw. das Einfügen in eine Persistent Queue durchgeführt oder abgebrochen wurde. Deshalb ist es sogar nach einem Systemzusammenbruch möglich, zu entscheiden, ob die Migration wiederholt werden muss. Ein Abbruch der Migration ohne Systemzusammenbruch stellt sowieso kein Problem dar, da am Returncode eines TRPC bzw. einer Queue-Operation erkannt werden kann, ob die Ausführung erfolgreich war.

Probleme, die im Zusammenhang mit dem zeitweisen oder dauerhaften Ausfall von WF-Servern auftreten können, sind unabhängig von der soeben diskutierten Fragestellung und müssen gesondert behandelt werden. Dazu können ähnliche Verfahren zur Erhöhung der Verfügbarkeit wie z.B. bei DBMS verwendet werden [ACPT99, CDK95, Len97, Web98], nämlich die Replikation von Daten und die Verwendung von Stand-By-Systemen. In [KAGM96] wird aufgezeigt, wie durch Verwendung solcher Verfahren ein hochgradig verfügbares WfMS realisiert werden kann.

3.3.2 Kommunikationszeitpunkte

Um ein effizientes verteiltes WF-Management zu realisieren, müssen die Anzahl der Kommunikationen und das übertragene Datenvolumen minimiert werden. Der erste Aspekt wird adressiert, indem untersucht wird, zu welchen Zeitpunkten (d.h. bei welchen Aktionen des WfMS) eine Kommunikation überhaupt erforderlich ist.

Bevor wir ein geeignetes Verfahren für ADEPT entwickeln, wollen wir zuerst die verteilte WF-Ausführung in MENTOR [MWW+98] betrachten, um von den dort gemachten Erfahrungen zu profitieren. MENTOR basiert auf der Partitionierung von State- und Activitycharts. Bei deren Ausführung laufen die Schritte paralleler Zweige quasi im „Gleichschritt“ ab.[3] Deshalb müssen diese nach der Ausführung jeder Aktivität synchronisiert werden. Um dabei eine zur zentralen Steuerung äquivalente WF-Ausführung zu erreichen, müssen nach Beendigung jeder Aktivität alle entstandenen Daten zwischen den parallelen Zweigen ausgetauscht werden. Diese *Strict Synchronization* [MWW+98] erfordert sehr viel Kommunikation. Dieses Modell kann verbessert werden (*Incremental Synchronization*), indem jeweils nur die Daten verschickt werden, die für die andere Partition relevant sind. Welche dies sind, kann zur Modellierungszeit ermittelt werden. Werden überhaupt keine Daten des

[3] Dieses Ausführungsmodell schränkt die Parallelität ein, da es Abhängigkeiten zwischen den Ausführungszeitpunkten von Aktivitäten paralleler Zweige erzwingt.

parallelen Zweiges benötigt, so muss trotzdem noch eine leere Nachricht zur Synchronisation verschickt werden. Es wird also nur die übertragene Datenmenge und nicht die Anzahl der Nachrichten reduziert. Deshalb wird in [MWW+98] auch noch ein sehr komplexes Verfahren entwickelt (*Weak Synchronization*), das ermittelt, ob eine Variable in einem bestimmten Zustand relevant ist und deshalb zwischen zwei Aktivitäten versandt werden muss. Dieses Verfahren kann aber zu semantischen Problemen führen und kann nur in einfachen Fällen auf Basis einer vom Modellierer manuell durchgeführten Zuordnung verwendet werden. Ansonsten muss die Incremental Synchronization verwendet werden. Das bedeutet, in den meisten praktisch relevanten Fällen sind sehr viele Kommunikationen notwendig. Dies gilt auch für parallele Zweige, zwischen denen keine Kanten verlaufen, so dass die Kommunikationspunkte nicht erkennbar sind. Ein solches Verhalten ist für ADEPT nicht erwünscht.

In ADEPT$_{distribution}$ sollen Kommunikationen ausschließlich entlang von Kanten des WF-Graphen stattfinden. Wie in Abb. 3.10 dargestellt, findet eine Migration genau dann statt, wenn der Server der Startaktivität einer Kante von dem der Zielaktivität verschieden ist. Im Folgenden wird eine Migration von Aktivität a zur Aktivität b als $M_{a,b}$ bezeichnet. Kommunikation zwischen den WF-Servern findet ausschließlich bei Migrationen statt. Da nur dann Information ausgetauscht wird, ist der Zustand von Aktivitäten paralleler Zweige i.Allg. nicht bekannt. So ist in Abb. 3.10 dem WF-Server 2 (z.B. beim Beenden der Aktivität c) nicht bekannt, in welchem Zustand sich die Aktivität e befindet. Er verfügt nur über die (evtl. veraltete) Information, dass Aktivität e noch nicht aktivierbar ist. Der Zustand von Aktivitäten paralleler Zweige wird für die Aktivitätenausführung aber auch nicht benötigt, weil für die Schalt- und Ausführungsregeln nur Vorgängeraktivitäten relevant sind und nur Daten gültig sind, die von Vorgängeraktivitäten geschrieben wurden. Es genügt also für die Steuerung der Aktivitäteninstanz a den Zustand der Vorgängeraktivitäten $PredInst^*_{\{CONTROL\_E,SYNC\_E,LOOP\_E\}}(a)$ zu kennen. In den nun folgenden Unterabschnitten wird für die verschiedenen Kantentypen untersucht, wann ihre Signalisierung eine Migration auslöst.

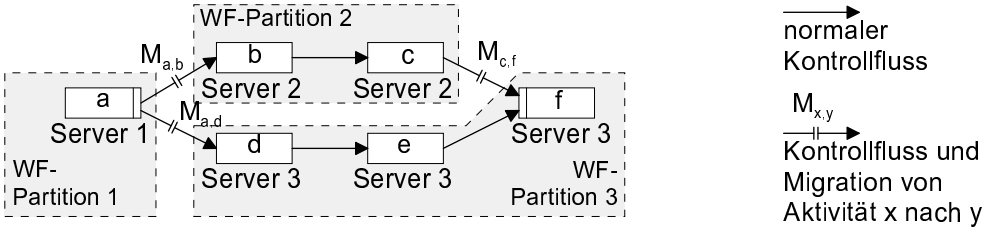


Abbildung 3.10 Beispiel aus Abb. 3.9 inklusive Migrationen.

3.3.2.1 Kontrollkanten

Entlang einer Kontrollkante von Aktivität m nach n findet natürlich nur dann eine Migration statt, wenn die Aktivitäten m und n von unterschiedlichen WF-Servern kontrolliert werden, d.h., wenn $Server_m \neq Server_n$ gilt. Ändert sich der Zustand der Kante aufgrund der in Abschnitt 2.2.2 beschriebenen Markierungsregeln von NOT_SIGNALED nach TRUE_SIGNALED, so muss entlang dieser Kante migriert werden. Danach wird die Bearbeitung der WF-Instanz vom neuen WF-Server mit der Nachfolgeraktivität fortgesetzt. So wird in Abb. 3.11b die Kante von Aktivität a nach b mit TRUE_SIGNALED markiert. Bevor die Aktivität b bearbeitet werden kann, muss entlang dieser Kante eine Migration stattfinden.

Im ADEPT-Modell werden Kontrollkanten mit FALSE_SIGNALED markiert, wenn sie einem Ausführungszweig einer exklusiven Verzweigung angehören, der nicht gewählt wurde. Entlang sol-

cher Kontrollkanten finden keine Migrationen statt. Auch dies wird an Abb. 3.11 erläutert: Nachdem die Verzweigungsentscheidung nach Ausführung der Aktivität a feststeht, werden die Kanten der nicht gewählten Zweige mit FALSE_SIGNALED und die entsprechenden Knoten mit SKIPPED markiert (der untere Zweig Abb. 3.11a). Danach wird die erste Kante des gewählten Zweiges mit TRUE_SIGNALED markiert (Kante $a \rightarrow b$ in Abb. 3.11b). Bevor die Migration zur Aktivität b stattfindet, wurden also die Kanten der nicht gewählten Zweige mit FALSE_SIGNALED markiert. Diese Kantenzustände werden bei der Migration weitergegeben und erreichen durch Migrationen entlang von Kanten mit dem Zustand TRUE_SIGNALED den Join-Knoten (Aktivität d in Abb. 3.11d). Es werden also keine Migrationen entlang von Kontrollkanten mit dem Zustand FALSE_SIGNALED ausgeführt. Algorithmisch wäre es zwar einfacher, auch Migrationen entlang von Kontrollkanten nicht gewählter Zweige durchzuführen. Dies würde aber einen unnötigen Aufwand bedeuten und steht damit im Widerspruch zu dem Ziel, ein effizientes WfMS zu realisieren.

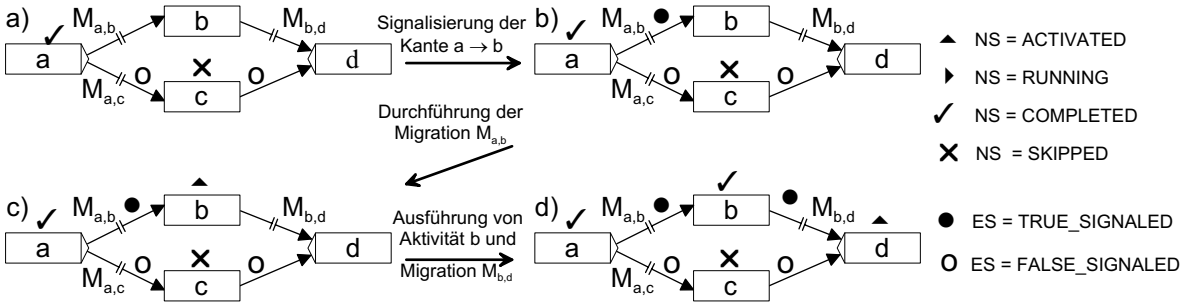


Abbildung 3.11 Ein Beispiel für Migrationen entlang von Kontrollkanten.

Um am Zustand einer WF-Instanz erkennen zu können, ob eine Migration schon durchgeführt wurde, muss die Menge der Kantenzustände erweitert werden. Prinzipiell könnte dazu ein weiterer Kantenzustand ES = MIGRATING eingeführt werden. Dies hat allerdings negative Seiteneffekte. So muss ein solcher zusätzlicher Zustand berücksichtigt werden, wenn z.B. dynamische Änderungen des Ablaufgraphen vorgenommen werden (siehe Kapitel 8 und [RD98, Rei00]). Deshalb wird eine andere Vorgehensweise gewählt. Die Kantenzustände ES und die Ausführungs- und Markierungsregeln bleiben gegenüber dem zentralen Fall unverändert. Dadurch werden Seiteneffekte auf andere Algorithmen vermieden. Zusätzlich zum Kantenzustand ES wird für jede Kante ein zusätzlicher Zustand MS (Migration State) eingeführt. Dieser ist initial NO_MIGRATION und kann auch die Werte BEFORE_MIGRATION und AFTER_MIGRATION annehmen. Der Gesamtzustand (Distributed Edge State) einer Kante ergibt sich als $DES = (ES, MS)$.

Nun bleibt noch zu klären, wann sich Zustandsänderungen von DES ergeben. Wie schon erwähnt wurde, ergeben sich Zustandsänderungen von ES aufgrund der Markierungsregeln des zentralen Falls (vgl. Abschnitt 2.2.2). MS einer Kontrollkante geht vom Zustand NO_MIGRATION in BEFORE_MIGRATION über, wenn ES von NOT_SIGNALED nach TRUE_SIGNALED wechselt und der Startserver der Kante von Zielserver verschieden ist. Nach erfolgreicher Beendigung der Migration geht MS beim Quell- und beim Zielserver der Migration in den Zustand AFTER_MIGRATION über. Dies erfolgt in derselben Transaktion wie die Migration, so dass der Zustand stets wiederspiegelt, ob die Migration erfolgreich verlaufen ist oder wiederholt werden muss (Zustand BEFORE_MIGRATION). Die Markierungsregeln DM_1 und DM_2 beschreiben diese Zustandsübergänge.

Markierungsregel DM_1 (Signalisierung von Kontrollkanten)
Wechselt der Zustand einer Kante $e = (m, n, CONTROL\_E)$ von $ES(e)$ = NOT_SIGNALED

nach $ES(e)$ = TRUE_SIGNALED und gilt $Server_m \neq Server_n$, so ändert sich $MS(e)$ in BEFORE_MIGRATION.

Markierungsregel DM_2 (Migration entlang von Kanten)
Wenn, nachdem alle Neuberechnungen der Kantenzustände ES und MS und der Knotenzustände NS abgeschlossen sind, sich eine Kante $e = (m, n, *^4)$ im Zustand $MS(e)$ = BEFORE_MIGRATION befindet, so wird eine Transaktion geöffnet, die WF-Instanz zum $Server_n$ migriert, bei erfolgreicher Migration der Zustand in MS = AFTER_MIGRATION geändert (bei $Server_m$ und $Server_n$) und die Transaktion beendet.

Die Ausführungsregeln des zentralen Falls bleiben unverändert, allerdings muss ihre Anwendbarkeit eingeschränkt werden. Eine Aktivität darf nur dann in den Zustand ACTIVATED wechseln (so dass sie gestartet werden kann), wenn sie aktuell vom richtigen Server kontrolliert wird und alle eingehenden Migrationen abgeschlossen sind (nicht Zustand BEFORE_MIGRATION[5]). Die Ausführungsregel DA_1 legt diese notwendigen Bedingungen für die Anwendung der Ausführungsregeln des zentralen Falls fest.

Ausführungsregel DA_1 (Anwendbarkeit von Ausführungsregeln)
Der Zustand einer Aktivität n mit den eingehenden Kanten $E = \{e \mid e = (*, n, *)\}$ kann auf dem WF-Server s nur dann von NS = NOT_ACTIVATED nach NS = ACTIVATED wechseln, wenn die folgenden Bedingungen erfüllt sind:
1) $Server_n = s$
2) $\forall e \in E : MS(e) \neq$ BEFORE_MIGRATION

3.3.2.2 Synchronisationskanten

Wird eine Synchronisationskante mit TRUE_SIGNALED markiert, so findet ebenso wie bei Kontrollkanten eine Migration statt, wenn der WF-Server der Startaktivität vom dem der Zielaktivität der Kante verschieden ist. Wird in dem in Abb. 3.12 dargestellten Beispiel die Aktivität d ausgeführt, so muss zum Server der Aktivität g migriert werden, da in g Daten gelesen werden können, die von der Vorgängeraktivität d geschrieben wurden. Anders als bei Kontrollkanten muss bei Synchronisationskanten auch dann eine Migration erfolgen, wenn diese mit FALSE_SIGNALED markiert wurden. Auch dies wird an Abb. 3.12 erläutert: Selbst wenn der obere Zweig der bedingten Verzweigung gewählt wird, kann die Aktivität g Daten lesen, die von b geschrieben wurden. Dies ist möglich, da b immer vor g ausgeführt wird, weil g erst gestartet werden kann, wenn die Synchronisationskante $d \rightarrow g$ (TRUE_SIGNALED oder FALSE_SIGNALED) signalisiert wurde. Dieser gesamte Sachverhalt wird durch die Markierungsregel DM_3 beschrieben. Die Markierungsregel DM_2 und die Ausführungsregel DA_1 gelten unverändert auch für Synchronisationskanten und wurden auch schon so formuliert, dass sie unabhängig vom Kantentyp sind.

Markierungsregel DM_3 (Signalisierung von Synchronisationskanten)
Wechselt der Zustand einer Kante $e = (m, n, SYNC\_E)$ von $ES(e)$ = NOT_SIGNALED nach

[4]Das Symbol * steht hier für einen beliebigen Wert.

[5]Der Zustand NO_MIGRATION muss nicht berücksichtigt werden, weil es für Kanten, die sich in diesem Zustand befinden, nur die folgenden beiden Möglichkeiten gibt: (1) Die Start- und Endeaktivität der Kante werden vom selben Server kontrolliert. Dann findet keine Migration statt, so dass in den Zustand ACTIVATED übergegangen werden darf. (2) Die beiden Aktivitäten werden von verschiedenen Servern kontrolliert. Dann ist der Zustand der Kante ES = NOT_SIGNALED, so dass ihre Endeaktivität sowieso noch nicht aktiviert werden kann. Die Kante kann sich nicht den Zustand ES = TRUE_SIGNALED befinden, weil dann die Markierungsregel DM_1 angewendet worden wäre. Dies hätte zu MS = BEFORE_MIGRATION geführt, was im Widerspruch zu der Annahme steht, dass MS = NO_MIGRATION gilt.

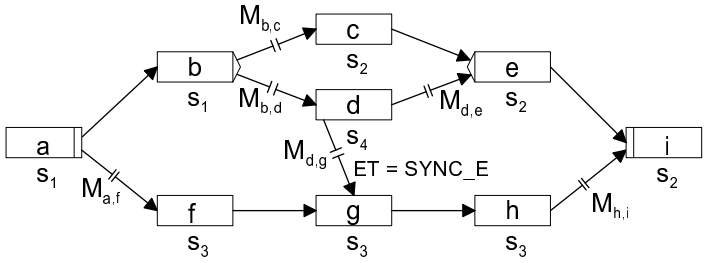


Abbildung 3.12 Ein Beispiel für Migrationen entlang von Synchronisationskanten.

$ES(e)$ = TRUE_SIGNALED oder nach $ES(e)$ = FALSE_SIGNALED und gilt $Server_m \neq Server_n$, so ändert sich $MS(e)$ in BEFORE_MIGRATION.

Durch die Anwendung der Markierungsregel DM_3 kann sich der „Migration State“ einer Synchronisationskante e in $MS(e)$ = BEFORE_MIGRATION ändern, was eine Migration auslöst. Bei einer Synchronisationskante e, welche von einem nicht gewählten Zweig einer bedingten Verzweigung ausgeht ($ES(e)$ = FALSE_SIGNALED), wird dieser Zustand vom Server der entsprechenden OR-Split-Aktivität ermittelt. Dieser Server führt auch die zugehörige Migration aus. Wird in dem Beispiel aus Abb. 3.12 der obere Zweig der bedingten Verzweigung ausgewählt, so führt der Server s_1 nach Beendigung der OR-Split-Aktivität b die Migration $M_{d,g}$ durch, weil s_1 die Markierungsregel DM_3 für die Synchronisationskante $d \rightarrow g$ anwendet. Die Migration $M_{d,g}$ geht also vom Quellserver s_1 aus, obwohl die Aktivität d eigentlich dem Server s_4 zugeordnet ist. Es ist aber wesentlich effizienter, die Daten direkt vom Server s_1 zum Server s_3 zu übertragen, anstatt sie zuerst zum Server s_4 und dann von diesem zum Server s_3 zu migrieren. Durch das gewählte Verfahren wird also unnötiger Kommunikationsaufwand vermieden.

3.3.2.3 Schleifenkanten

Wird ein Schleifenendeknoten (Aktivität d in Abb. 3.13a) beendet, so wird abhängig von einem Datenelement die ausgehende Kontrollkante oder die Schleifenkante ($d \rightarrow a$) mit TRUE_SIGNALED markiert. Trifft dies auf die Kontrollkante zu, so findet entlang der Schleifenkante natürlich keine Migration statt, da die Schleife verlassen wurde. Wird die Schleifenkante mit TRUE_SIGNALED markiert, so werden alle Markierungen des Schleifenrumpfs zurückgenommen (vgl. Abb. 3.13b). Bevor mit der erneuten Ausführung der Schleife begonnen werden kann, muss zum Startknoten der Schleife migriert werden. Da bei Schleifenkanten also nur migriert werden muss, wenn sich deren Zustand in TRUE_SIGNALED ändert, ergibt sich dasselbe Verhalten wie bei Kontrollkanten. Dies wird durch die auf Schleifenkanten erweiterte Markierungsregel DM_1 berücksichtigt. Die Markierungsregel DM_2 und die Ausführungsregel DA_1 gelten auch für Schleifenkanten unverändert.

Markierungsregel DM_1 (Signalisierung von Kontroll- und Schleifenkanten)
Wechselt der Zustand einer Kante $e = (m, n, edgeType)$ mit $edgeType \in \{CONTROL\_E, LOOP\_E\}$ von $ES(e)$ = NOT_SIGNALED nach $ES(e)$ = TRUE_SIGNALED und gilt $Server_m \neq Server_n$, so ändert sich $MS(e)$ in BEFORE_MIGRATION.

3.3.3 Migration von Zustandsinformation

Nachdem geklärt wurde, wann eine Migration stattfindet, wird in diesem Abschnitt untersucht, wie die Zustandsinformation einer WF-Instanz (WF Control Data nach [WMC99]) migriert werden soll. Dazu

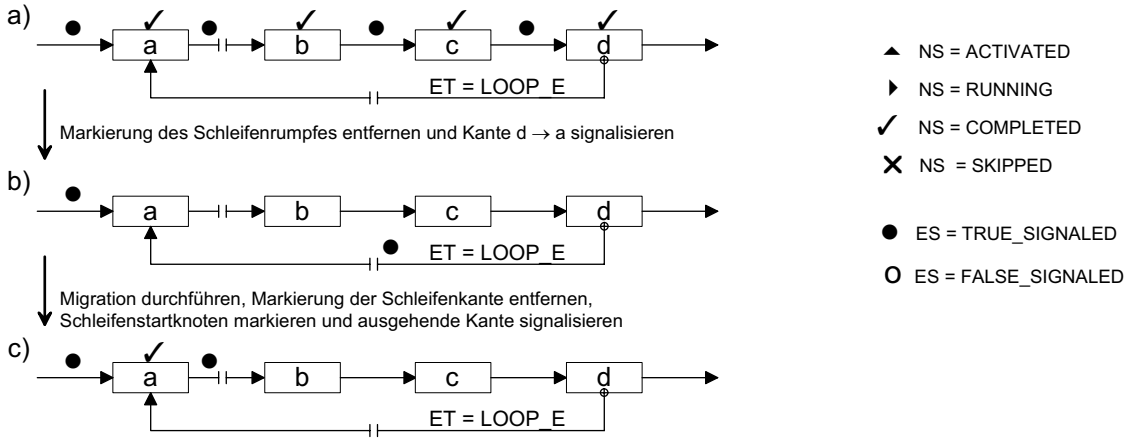


Abbildung 3.13 Ein Beispiel für eine Migration entlang einer Schleifenkante.

werden die prinzipiell möglichen Vorgehensweisen analysiert, ein geeignetes Verfahren ausgewählt und näher beschrieben. In Kapitel 6 wird untersucht, wie dieses Verfahren noch optimiert werden kann. Die Migration von WF-relevanten Daten und Anwendungsdaten wird jetzt noch ausgeklammert und stattdessen in Abschnitt 3.3.4 betrachtet.

3.3.3.1 Auswahl eines geeigneten Verfahrens

Im Folgenden werden mögliche Ansätze zur Migration der Zustandsinformation einer WF-Instanz diskutiert. Dabei wird außer der explizit aufgeführten Information stets auch noch die notwendige Kontextinformation (die WF-Instanz-ID) transferiert.

Ansatz 1: Minimallösung
Die Übertragung der Zustandsinformation ist mit minimalem Kommunikationsaufwand möglich, wenn dem Zielserver lediglich die Quell- und Zielaktivität der Migration mitgeteilt wird. Damit weiß er, um welche Migration es sich handelt, und somit, an welcher Stelle im WF-Ausführungsgraphen er die Abarbeitung fortsetzen muss.
Der Nachteil dieses einfachen Ansatzes ist das Fehlen jeglicher Zusatzinformation aus der Ablaufhistorie. So ist nicht bekannt, von welchen Benutzern die Vorgängeraktivitäten bearbeitet wurden, was die Verwendung von abhängigen Bearbeiterzuordnungen ausschließt. Auch sind keine Ausführungszeitpunkte bekannt, weshalb z.B. Zeitbedingungen nicht überwacht werden können [Gri97].

Ansatz 2: Übertragung der aktuellen Zustände
Um das Problem der fehlenden Zustandsinformation zu lösen, können bei einer Migration alle Zustände der Aktivitäten und Kanten des Ausführungsgraphen an den Migrationszielserver übertragen werden. Bei Synchronisationspunkten (z.B. Zusammenführung paralleler Zweige) werden von den verschiedenen Migrationsquellservern i.d.R. unterschiedliche Zustandsinformationen zu denselben Aktivitäten geliefert, da diese für nicht von ihnen selbst kontrollierte Aktivitäten evtl. nur veraltete Zustandsinformation besitzen. Damit stellt sich das Problem, zu entscheiden, welches der aktuelle Zustand einer Aktivität ist, da dies nicht immer der am weitesten fortgeschrittene Zustand sein muss. So kann z.B. beim Zurücksetzen der Zustand einer Aktivität von TERMINATED nach NOT_ACTIVATED übergehen. Werden nun bei verschiedenen Migrationen unterschiedliche Zustände für diese Aktivität empfangen, so kann nicht ohne weiteres entschieden werden, welches der aktuell gültige ist.

Zwar ist bei diesem Ansatz die Zustandsinformation verfügbar, allerdings fehlt auch hier Zusatzinformation zu Start-/Endezeitpunkten oder den Bearbeitern von Aktivitäteninstanzen. Dies führt wieder zu den bei Ansatz 1 beschriebenen Problemen.

Ansatz 3: Übertragung der Zustände und der Ablaufhistorie
Um das Problem fehlender Zusatzinformation zu lösen, kann die gesamte zu dieser WF-Instanz gespeicherte Information (inkl. Ablaufhistorie) übertragen werden. Werden an Synchronisationspunkten unterschiedliche Zustände für eine Aktivität empfangen, so muss wie beim Ansatz 2 der aktuellste Zustand bestimmt werden.
Da bei diesem Ansatz alle Kontrolldaten der WF-Instanz migriert werden, treten die oben beschriebenen Probleme nicht auf. Der Nachteil ist aber, dass Ausführungsinformationen redundant übertragen werden, da sich z.B. die Information, dass eine Aktivität beendet wurde, sowohl in der Ablaufhistorie als auch in den Aktivitätenmarkierungen widerspiegelt.

Ansatz 4: Übertragung (nur) der Ablaufhistorie
Die gesamte von Nachfolgeraktivitäten benötigte Information zu einer WF-Instanz findet sich in deren Ablaufhistorie wieder. In dieser sind sowohl die Bearbeiter wie auch die Start- und Endezeitpunkte von Aktivitäten vermerkt. Ausgehend von der (repliziert vorhandenen) WF-Vorlage kann der Zustand der WF-Instanz durch das „Nachspielen“ der Ablaufhistorie und die Anwendung der Ausführungs- und Markierungsregeln (Abschnitt 2.2.2) rekonstruiert werden. Deshalb kann die Zustandsinformation einer WF-Instanz auch übertragen werden, indem ausschließlich die Ablaufhistorie transferiert wird.

Bewertung
Ein gravierender Nachteil der Ansätze 1 und 2 ist das Fehlen von Zusatzinformation zur WF-Instanz. Sollen bei Zugrundelegung dieser Ansätze fortschrittliche WfMS-Konzepte realisiert werden, muss häufig Information nachträglich angefordert werden. Dies führt dazu, dass eine große Anzahl von Übertragungen stattfindet, da die Information, die bei den anderen Ansätzen auf einmal übertragen wird, nun durch mehrere Anforderungen besorgt werden muss. Im Extremfall müssen von allen Vorgängeraktivitäten Daten nachgefordert werden, z.B. wenn die Start- und Endezeitpunkte aller Aktivitäten benötigt werden, um Zeitpläne für die nachfolgenden Aktivitäten zu berechnen (vgl. [Gri97, DRK00]). Das Nachfordern beeinträchtigt außerdem die Verfügbarkeit des WfMS, da die Bearbeitung eines WF nur fortschreiten kann, wenn die Server, von denen Informationen benötigt werden, gerade verfügbar sind. Aus diesen Gründen scheiden die Ansätze 1 und 2 aus.[6] Der Ansatz 3 disqualifiziert sich wegen des großen Umfangs der zu transferierenden Datenmenge. Abgesehen davon resultiert aus den zusätzlich übertragenen Daten kein Vorteil gegenüber dem Ansatz 4. Der letztgenannte Ansatz stellt einen guten Kompromiss dar, da hier die Datenmenge in einem vernünftigen Rahmen bleibt und alle benötigte Information vorhanden ist. Im Folgenden wird dieser Ansatz deshalb näher untersucht.

3.3.3.2 Migration der Ablaufhistorie

Nachdem sich gezeigt hat, dass die günstigste Variante zur Migration von Zustandsinformation ist, die Ablaufhistorie zu übertragen, betrachten wir nun diesen Ansatz etwas detaillierter. Insbesondere

[6] In Abschnitt 6.2 wird ein Verfahren vorgestellt, das mit der Übertragung derselben Information wie der Ansatz 1 beginnt (WF-Instanz-ID, Quell- und Zielaktivität der Migration). Anschließend wird die zusätzlich benötigte Information angefordert und (auf einmal) übertragen. Da das Verfahren auf der Übertragung von Ablaufhistorien basiert, wird es aber als Optimierung des Ansatzes 4 betrachtet.

ist interessant, wie an Synchronisationspunkten die Ablaufhistorien der verschiedenen Vorgängeraktivitäten zusammengeführt werden können.

Bei einem WF ohne parallele Verzweigungen ist der Ablauf des Verfahrens denkbar einfach: Da eine Ablaufhistorie stets von nur einer Vorgängeraktivität empfangen wird, kann die lokal evtl. schon vorhandene Historie durch diese ersetzt werden. Durch das „Nachspielen“ der in der Ablaufhistorie vorhandenen Einträge und die Anwendung der Ausführungs- und Markierungsregeln aus Abschnitt 2.2.2 erhält man dann die aktuelle Markierung des WF-Graphen.

Interessanter ist der Fall, dass sich von unterschiedlichen Servern kontrollierte parallele Zweige an einem Punkt synchronisieren, da dann zwei unterschiedliche Versionen der Ablaufhistorie aufeinander treffen. Das Zusammenführen der Information kann bei einem AND-Join-Knoten erforderlich werden, aber auch innerhalb einer Parallelität bei Knoten mit einmündenden Synchronisationskanten. Im Beispiel aus Abb. 3.14 kommen beide Fälle vor, und zwar bei der Migration $M_{b,e}$ über die Synchronisationskante zwischen Aktivität b und e und bei der Migrationen zum Join-Knoten f. Beim Zusammenführen von Historieninformation verschiedener WF-Server muss gewährleistet sein, dass das Ergebnis wieder eine korrekte Historie darstellt (siehe Definition 3.4). Es müssen sich also alle im WF-Graphen definierten Reihenfolgebeziehungen von Aktivitäten in der Ablaufhistorie widerspiegeln, auch wenn diese Aktivitäten von unterschiedlichen Servern kontrolliert wurden. Des Weiteren muss ein WF-Server stets den vollständigen für ihn relevanten Zustand einer von ihm kontrollierten WF-Instanz kennen. Das heißt, ein WF-Server, der gerade die Aktivitäteninstanz a kontrolliert, kennt stets alle Historieneinträge (und damit den Zustand) der Vorgängeraktivitäten von a, weil diese bei Migrationen zu ihm weitergereicht wurden. Über die Historieneinträge von parallel zu a ausgeführten Aktivitäten muss dieser Server im Allgemeinen aber nicht verfügen.

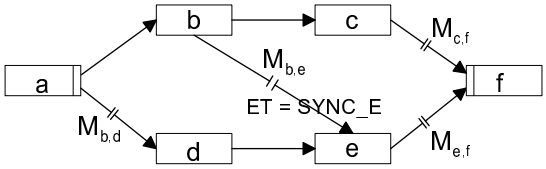


Abbildung 3.14 Synchronisationsstellen beim Zusammenführen von Ablaufhistorien.

Definition 3.4 (Korrektheit von Ablaufhistorien bei verteilter Ausführung)
Eine Ablaufhistorie ist dann korrekt, wenn sie auch auf einem zentralen System mit nur einem einzigen WF-Server entstanden sein könnte. Zum zentralen Fall darf jedoch der Unterschied bestehen, dass in der Historie für die verschiedenen Aktionen unterschiedliche WF-Server vermerkt sind.

Das Zusammenführen der Ablaufhistorien (History Merge) funktioniert folgendermaßen (vgl. Algorithmus 3.1). Die erste empfangene oder schon lokal vorhandene Historie H_1 wird unverändert übernommen und bildet den Anfang der Ergebnishistorie H. Die Einträge E der anderen Historie H_2 werden in der Reihenfolge durchlaufen, wie sie in H_2 aufgeführt sind. Einträge E, die in H_1 nicht enthalten sind, werden an H angehängt.

Wir wollen den Algorithmus zuerst am Beispiel aus Abb. 3.14 betrachten. In einer korrekten Ablaufhistorie H müssen die Einträge zur Aktivität b stets vor den Einträgen zur Aktivität e auftreten.[7] Zunächst betrachten wir die Migration $M_{b,e}$, die nach dem Signalisieren der Synchronisationskante

[7] Bei dieser Aussage wird angenommen, dass der Graph-Ausschnitt nicht in einer Schleife liegt. Anderenfalls müsste die Betrachtung auf jeweils eine Schleifeniteration beschränkt werden.

$b \rightarrow e$ stattfindet. Die Aktivität e kann erst ausgeführt werden, wenn die Aktivität b beendet ist. Die Migration $M_{b,e}$ und das Zusammenführen der Ablaufhistorien (inklusive der Einträge zu b) findet vor der Ausführung der Aktivität e statt. Einträge zur Aktivität b befinden sich deshalb immer vor Einträgen zur Aktivität e. Die Reihenfolge von Einträgen der Aktivitäten b und d spielt keine Rolle, da die Graphstruktur keine Reihenfolgebeziehung zwischen diesen Aktivitäten vorgibt. Deshalb ist es korrekt, dass die in H_2 enthaltenen Einträge der Aktivität b hinter den Einträgen zu d aus H_1 angehängt werden.

Betrachten wir nun das Zusammenführen der Ablaufhistorien an der Join-Aktivität f: Stark vereinfacht werden bei der Migration $M_{c,f}$ die Einträge $[a, b, c]$ und bei $M_{e,f}$ die Einträge $[a, d, b, e]$ übertragen. Wird die Migration $M_{c,f}$ zuerst ausgeführt, so ergibt sich H durch Übernahme von $H_1 = [a, b, c]$ und Anhängen der restlichen Einträge $[d, e]$ aus $H_2 = [a, d, b, e]$) als $H = [a, b, c, d, e]$. Dies ist korrekt, da alle durch die Graphstruktur vorgegebenen Reihenfolgebeziehungen berücksichtigt sind. Wird zuerst die Migration $M_{e,f}$ ausgeführt, so wird $H_1 = [a, d, b, e]$) in H übernommen und der noch nicht enthaltene Eintrag $[c]$ aus $H_2 = [a, b, c]$ wird angehängt. Damit ergibt sich die korrekte Ablaufhistorie $H = [a, d, b, e, c]$).

Algorithmus 3.1 (Mischen von Ablaufhistorien)
input
H_1: schon vorhandene oder zuerst eingetroffene Historie
H_2: die zu integrierende Historie
output
H: die durch Mischen von H_1 und H_2 erzeugte Historie
begin
$H = H_1$;
for each $E \in H_2$ in der Reihenfolge aus H_2 **do**
if $E \notin H_1$ **then**
H = Append(H, E);
end.

Satz 3.1 (Zusammenführung von zwei Ablaufhistorien)
Wenn 2 Ablaufhistorien H_1 und H_2, die nach Definition 3.4 korrekt sind, mit Algorithmus 3.1 gemischt werden, so entsteht wieder eine korrekte Ablaufhistorie H. Bei diesem Vorgang gehen keine Einträge aus H_1 oder H_2 verloren.

Beweis zu Satz 3.1:
Dass in der Ergebnishistorie H alle Einträge vorhanden sind, die in H_1 oder H_2 enthalten sind, ist aufgrund der Funktionsweise von Algorithmus 3.1 klar. Es bleibt nur noch zu zeigen, dass die Reihenfolge der Einträge korrekt ist. Insbesondere muss gezeigt werden, dass Einträge aus H_2 einfach an die Einträge aus H_1 angehängt werden dürfen und nie zwischen Einträgen aus H_1 einsortiert werden müssen.

Die Reihenfolge der Einträge $E \in H_1$ in H ist sicherlich korrekt, da H_1 korrekt ist und die Einträge bzgl. ihrer Reihenfolge unverändert in H übernommen werden. Außerdem ist die Reihenfolge der Einträge E, welche nur in H_2 vorkommen, korrekt, weil diese in unveränderter Reihenfolge an H_1 angehängt werden. Es bleibt also noch zu zeigen, dass auch die Reihenfolgebeziehung zwischen Einträgen, die in H_1 vorkommen, und solchen, die ausschließlich in H_2 vorkommen, korrekt ist:
Betrachten wir nun einen beliebigen Historieneintrag E', der ausschließlich in H_2 vorkommt: $E' \in$

$H_2 \wedge E' \notin H_1$
Betrachten wir außerdem einen Eintrag E, der in H_1 vorkommt: $E \in H_1$
Dann gilt: $\forall E', \forall E:$ $ActivityInstance(E')$ ist Nachfolger von $ActivityInstance(E) \vee$
$ActivityInstance(E')$ wird parallel zu $ActivityInstance(E)$ ausgeführt
Dies ist genau dann der Fall, wenn: $\nexists E', E$ mit
$ActivityInstance(E') \in PredInst^*_{\{CONTROL\_E, SYNC\_E, LOOP\_E\}}(ActivityInstance(E))$
Dies wird durch Widerspruch gezeigt: Wenn $ActivityInstance(E')$ eine Vorgängeraktivität von $ActivityInstance(E)$ ist und der Eintrag E befindet sich in H_1, so existiert der Eintrag E' auch in H_1 (in einer Historie befinden sich stets auch die Einträge zu allen Vorgängeraktivitäten der enthaltenen Aktivitäten, da bei der Ausführung dieser Aktivität die Information über die Vorgängeraktivitäten benötigt wird). Dies steht im Widerspruch zu der Anforderung an den Eintrag E', dass $E' \notin H_1$ gilt. Da $ActivityInstance(E')$ nach oder parallel zu $ActivityInstance(E)$ ausgeführt wird, ist es korrekt, dass der Eintrag E' in der Ablaufhistorie H nach dem Eintrag E eingereiht wird. □

Satz 3.2 (Zusammenführung von mehr als zwei Ablaufhistorien)
Wenn die Ablaufhistorien $H_1 \ldots H_n$ ($n > 2$), die nach Definition 3.4 korrekt sind, sukzessive (d.h. erst H_1 mit H_2, dann das Ergebnis mit H_3 usw.) mit Algorithmus 3.1 gemischt werden, so entsteht wieder eine korrekte Ablaufhistorie H.

Beweis zu Satz 3.2:
Satz 3.2 lässt sich leicht durch vollständige Induktion zeigen:
Induktionsvoraussetzung: Die sukzessive Anwendung von Algorithmus 3.1 führt bei k Eingangshistorien zu einer korrekten Historie H.
Induktionsanfang ($k = 2$): Nach Satz 3.1 werden 2 Historien durch Algorithmus 3.1 korrekt zusammengeführt.
Induktionsschritt ($k \rightarrow k + 1$): Die durch Mischen von k Historien entstandene Ablaufhistorie H ist nach Induktionsvoraussetzung korrekt (bzgl. Definition 3.4). Das Mischen einer korrekten Ablaufhistorie H mit einer weiteren Historie führt nach Satz 3.1 wieder zu einer korrekten Historie, die jetzt die Historien $H_1 \ldots H_{k+1}$ beinhaltet. □

Um auf dem Zielserver der Migration die aktuellen Graphmarkierungen zu erhalten, müssen die Einträge der Ablaufhistorien „nachgespielt“ werden. Häufig ist die durch H_1 determinierte Markierung aber schon bekannt, weil die Vorgängeraktivität einer Synchronisationsaktivität ebenfalls vom aktuellen Server kontrolliert wurde. Dann genügt es den Teil $H - H_1$ der durch Algorithmus 3.1 erzeugten Ablaufhistorie H „nachzuspielen“, wobei von der bekannten Graphmarkierung ausgegangen wird. Diese Optimierung ist möglich, weil die Historie H_1 von Algorithmus 3.1 unverändert übernommen wird und die neuen Einträge am Ende der Ablaufhistorie angehängt werden.

Mit einer verwandten Aufgabenstellung beschäftigt sich das Projekt TransCoop [WK96] in Bezug auf Concurrency Control beim verteilten Editieren komplexer Dokumente: Zwei oder mehr Personen arbeiten – zumindest zeitweise – am selben Dokument. Dadurch entstehen mehrere gültige Änderungshistorien. Um daraus wieder ein gemeinsames Dokument zu erzeugen, ist ein Verfahren zum Mischen der Änderungshistorien notwendig, so dass wieder eine gültige Änderungshistorie entsteht. Es werden formale Bedingungen für eine korrekte Änderungshistorie aufgestellt und schließlich ein Algorithmus zum Import von Teilhistorien beschrieben. Die Autoren betonen, dass es wichtig ist, die anwendungsbezogene Semantik auszunutzen. Ähnliches geschieht auch beim Algorithmus 3.1, indem Wissen über die Reihenfolge der Erzeugung der Historieneinträge auf den verschiedenen Servern genutzt wird, um die Ablaufhistorien an Synchronisationspunkten wieder korrekt zusammenzuführen.

3.3.4 Migration von Datenelementen

Nachdem geklärt wurde, wie die Zustandsinformation bei einer Migration übertragen wird, wird dies nun für die Datenelemente (Application Data und WF Relevant Data nach [WMC99]) geklärt. Dazu ist zu untersuchen, in welchem Format ein Datenelement übertragen werden muss und welche Varianten es für die Migration eines Datenelements gibt. In Kapitel 6 werden noch Optimierungen des gewählten Verfahrens diskutiert.

Wie schon in Abschnitt 2.3.2.1 erläutert wurde, enthält ein Datenhistorieneintrag die ID des Datenelements, dessen Wert und die ID der Aktivitäteninstanz, welche diesen Wert geschrieben hat. Zusätzlich zu diesen Informationen wird bei der Migration von Datenelementen als Kontextinformation die ID der migrierten WF-Instanz übertragen. Im Folgenden werden die für die Übertragung von Datenelementen möglichen Varianten diskutiert:

1. Für jedes Datenelement existiert eine Historie aller während der Ausführung der WF-Instanz angenommenen Werte. Diese wird benötigt, um nach einem Zurücksetzen die Bearbeitung mit den korrekten Eingabeparametern für die Aktivitätenprogramme fortsetzen zu können. Bei einer Migration kann die komplette Historie der Datenelemente übertragen werden.

   Dieses Verfahren kann bei häufig veränderten Datenelementen zu einem sehr großen Datenvolumen führen.

2. Für die Startaktivität einer Migration ist nur eine Version eines Datenelements gültig (die zuletzt von einer Vorgängeraktivität geschriebene Version). Nur diese wird von den Nachfolgeraktivitäten potentiell benötigt. Deshalb genügt es, nur diese „aktuelle Version“ des Datenelements zu migrieren. Alte Werte des Datenelements werden nur beim Zurücksetzen benötigt. Dieser Fall wird in Abschnitt 3.3.5 diskutiert.

   Am Beispiel aus Abb. 3.15 wird erläutert, welche Version eines Datenelements migriert werden muss und warum es ausreicht, jeweils die aktuell gültige Version zu migrieren. Bei der Migration $M_{f,i}$ ist am Startknoten f der Migration der von Aktivität b geschriebene Wert des Datenelements x gültig. Deshalb muss dieser Wert transferiert werden. Selbst wenn die Aktivität c schon eine neue Version von x erzeugt hat, muss diese nicht übertragen werden, da c kein Vorgänger von f ist. Bei der Migration $M_{g,j}$ sind die von den Aktivitäten b und c geschriebenen Werte von x nicht mehr gültig, da sie von Aktivität g überschrieben wurden. Deshalb muss die von g erzeugte Version von x migriert werden. Bei der Migration $M_{i,j}$ wird der von b geschriebene Wert von x migriert, da dies die im unteren Zweig gültige Version ist. In diesem Beispiel wird die Aktivität j den bei der Migration $M_{g,j}$ erhaltenen Wert von x lesen, da dieser aktueller ist (vgl. Abschnitt 2.3.2.1).

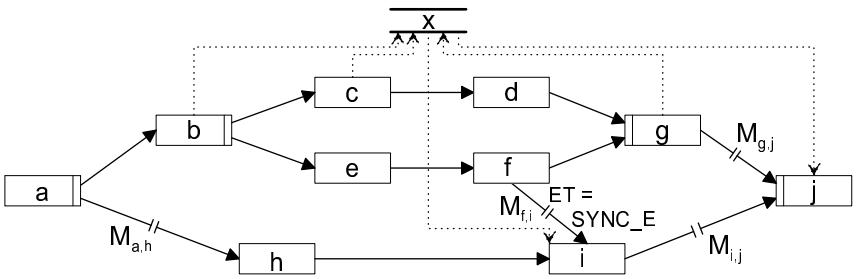


Abbildung 3.15 Beispiel für die Migration eines Datenelements.

Das 2. vorgestellte Verfahren erfordert bei Migrationen die Übertragung einer wesentlich kleineren

Datenmenge. Außerdem hat es bei der Vorwärtsausführung von WF-Instanzen keinerlei Nachteile. Deshalb wird dieses Verfahren gewählt.

Im Gegensatz zu den Ablaufhistorien (siehe Abschnitt 3.3.3) ist bei den Datenelementen kein Algorithmus zum Mischen an Synchronisationspunkten notwendig. Schon im zentralen Fall sind mehrere Versionen eines Datenelements bekannt (Historie des Datenelements). Basierend auf Regeln für den Datenfluss wird die jeweils aktuellste gültige Version gelesen. Der Unterschied im verteilten Fall ist lediglich, dass nicht alle Version eines Datenelements bekannt sind. Die aktuell gültige ist aber stets vorhanden, da von allen Vorgängeraktivitäten die jeweils gültige Version übertragen wird (bzw. schon lokal vorhanden ist, falls die Vorgängeraktivität vom selben Server kontrolliert wird).

3.3.5 Migration beim Zurücksetzen

Das im vorherigen Abschnitt beschriebene Verfahren zur Übertragung von Datenelementen bei Migrationen hat Auswirkungen auf das Zurücksetzen von WF-Instanzen, da einem WF-Server nicht mehr alle Versionen eines Datenelements bekannt sind. Der Vorteil des gewählten Verfahrens war, dass die normale WF-Ausführung (Vorwärtsausführung) effizient realisiert werden kann. Dafür muss beim vergleichsweise seltenen Fall des Zurücksetzens ein erhöhter Aufwand akzeptiert werden. In diesem Abschnitt wird untersucht, wie es dennoch möglich ist, das Zurücksetzen effizient zu realisieren.

Beim Abbruch einer Aktivität wird ihr Zustand von RUNNING auf ACTIVATED zurückgesetzt. Dieser Fall ist unkritisch, da die dafür notwendigen Aktionen auf einem einzelnen WF-Server stattfinden, so dass dies unabhängig von Verteilungsaspekten ist. Aufgrund vordefinierter Bedingungen oder wegen einer expliziten Anforderung eines Benutzers (mit entsprechenden Rechten), kann in ADEPT aber auch bis zu einer beliebigen Vorgängeraktivität zurückgesetzt werden. Vereinfacht gesprochen müssen dann alle dazwischen liegenden Aktivitäten kompensiert werden, bevor die Ausführung der WF-Instanz fortgesetzt werden kann. Das Zurücksetzen kann also einen größeren Bereich des WF-Graphen betreffen, in dem sich auch Migrationskanten befinden können. Das Zurücksetzen von WF und Transaktionen, welche mehrere Aktivitäten umspannen, sind ein eigenes Forschungsgebiet [AAE+96, KMO98, KR96, Ley95, Ley97, RS94, SR93, WS97] und nicht zentrales Anliegen dieser Arbeit. Deshalb wurden diese Fragestellungen nur in soweit untersucht, wie sie die verteilte WF-Ausführung betreffen. Die Auswirkungen der beim Rücksetzen notwendigen Aktionen werden bei der Bestimmung der optimalen Verteilung (also der Serverzuordnungen) ignoriert, da die Kosten und die Häufigkeit von Rücksetzoperationen schwer abschätzbar sind und außerdem (wie schon erwähnt) der „Normalfall“ optimiert werden soll.

Die Rekonstruktion des nach dem Zurücksetzen entstehenden Zustandes der WF-Instanz ist unproblematisch, da – wie im zentralen Fall – dem aktuellen Server die gesamte Ablaufhistorie bekannt ist: Das Zurücksetzen wird in der Ablaufhistorie vermerkt und dieselben Regeln wie im zentralen Fall (siehe [Rei00]) werden auf den Zustand der WF-Instanz angewandt, um den durch das Zurücksetzen resultierenden Zustand zu ermitteln.

Wir wollen nun untersuchen, wie das Zurücksetzen im verteilten Fall abläuft: Um eine Kompensationsaktivität starten zu können, werden die alten Versionen der Datenelemente benötigt, weil eine Kompensationsaktivität $\overline{a}$ i.Allg. diejenigen Daten benötigt, die auch von der zu kompensierenden Aktivität a verwendet wurden. Außerdem werden diese Daten für die erneute Ausführung der Aktivität a, nach Abschluss der Kompensation, benötigt. Das Problem ist nun, dass die alten Werte eines Datenelements bei einer Migration nicht zum Zielserver übertragen werden (vgl. Abschnitt 3.3.4), weshalb sie den Servern nachfolgender Aktivitäten nicht bekannt sind. Da derjenige Server, der die

Aktivität a ursprünglich ausgeführt hat, über diese Daten verfügt, kann aber dieser die Kompensationsaktivität $\overline{a}$ ausführen. Dieser Server ist dafür auch gut geeignet, da er eine hohe Lokalität zu den Bearbeitern von a aufweist, und damit häufig auch zu denen von $\overline{a}$. Das soeben beschriebene Verhalten wird durch das folgende Vorgehen erreicht: Beim Zurücksetzen wird die Ablaufhistorie rückwärts durchlaufen. Dabei werden die Aktivitäten kompensiert. Migrationen werden in umgekehrter Richtung ausgeführt, d.h. bei dieser „Rückwärtsmigration“ muss der Quell- und der Zielserver gegenüber der „Originalmigration“ vertauscht werden. Nach Abschluss des Zurücksetzens befindet sich die WF-Instanz an demjenigen WF-Server, der die als letzte kompensierte Aktivität ursprünglich ausgeführt hatte. Dieser verfügt über die notwendigen Eingabedaten für das Aktivitätenprogramm und ist auch für dessen Steuerung vorgesehen. Deshalb kann er mit der Vorwärts-Ausführung der WF-Instanz fortfahren.

Wir wollen nun noch eine alternative Vorgehensweise betrachten: Der WF-Server, bei dem das Zurücksetzen initiiert wird, kann alle Kompensationsaktivitäten $\overline{a}$ selbst starten. Da er nicht über die dafür notwendigen alten Versionen der Datenelemente verfügt, muss er diese erst beim entsprechenden Server anfordern. Nach Abschluss des Zurücksetzens stößt er eine Migration zu demjenigen WF-Server an, bei dem die Abarbeitung der WF-Instanz fortgesetzt werden soll. Diese Variante kommt immer mit dieser einen Migration aus, während beim vorher vorgestellten Verfahren mehrere Migrationen notwendig werden können. Nachteile der Alternative sind aber, dass Daten extra angefordert werden müssen, was eigentlich ebenfalls einer Migration entspricht, und dass die Kompensationsaktivitäten u.U. von einem äußerst ungünstigen WF-Server gesteuert werden. Aufgrund dieser Nachteile wird diese Alternative nicht weiter verfolgt.

Das Zurücksetzen kann also so realisiert werden, dass dafür während der normalen WF-Ausführung keine Vorkehrungen getroffen werden müssen. Lediglich der Zustand einer WF-Instanz und die alten Versionen von Datenelementen müssen aufbewahrt werden, selbst wenn die WF-Instanz von dem entsprechenden Server nicht mehr kontrolliert wird. Diese können erst gelöscht werden, wenn der WF beendet wurde. Beim Zurücksetzen wird eine Rückwärts-Migration zu demjenigen Server notwendig, der die zu kompensierende Aktivität kontrolliert hat. Dieser ist i.d.R. auch gut zur Steuerung der Kompensationsaktivität geeignet. Außerdem kann diese Rückwärts-Migration effizient realisiert werden (vgl. Abschnitt 6), da der Zielserver schon einen Großteil der Ablaufhistorie der WF-Instanz und alle vor der Migration gültigen Versionen der Datenelemente kennt. Deshalb muss bei dieser Rückwärts-Migration nur noch ein kleiner Teil der Ablaufhistorie übertragen werden. Eine Übertragung von Datenelementen ist nicht notwendig.

3.4 Alternative Vorgehensweisen

In den nun folgenden Unterabschnitten werden Alternativen zur logischen Partitionierung der WF-Vorlagen bei deren Modellierung und den daraus resultierenden Migrationen aufgezeigt. Außerdem wird erläutert, weshalb wir uns in ADEPT$_{distribution}$ gegen diese entschieden haben.

3.4.1 Festlegung der Serverzuordnungen zu Ausführungszeit

Eine Alternative zur Festlegung von Serverzuordnungen zur Modellierungszeit ist, die WF-Server dynamisch zur Ausführungszeit der WF-Instanzen festzulegen. Nach Beendigung jeder Aktivität muss dann für die Nachfolgeraktivität(en) der am besten geeignete WF-Server bestimmt werden. Zu diesem Zeitpunkt ist für diese Entscheidung ein Maximum an Information verfügbar. Da wir in diesem

Kapitel annehmen, dass fast ausschließlich unabhängige Bearbeiterzuordnungen verwendet werden,[8] ist die Menge der potentiellen Bearbeiter einer Aktivität statisch. Diese Menge ist also schon zur Modellierungszeit bekannt. Da deshalb zur Ausführungszeit keine zusätzliche Information zur Verfügung steht, bringt die dynamische Festlegung des WF-Servers keine Vorteile. Sie hat aber einen gravierenden Nachteil: Die Berechnung des optimalen WF-Servers für die Nachfolgeraktivität(en) erfordert einen großen Aufwand, da komplexe Analysen durchgeführt werden müssen. Diese sind ähnlich zu den in Kapitel 4 beschriebenen Verteilungsalgorithmen. Die Analysen müssen aber zur Ausführungszeit von einem ohnehin schon stark belasteten WF-Server durchgeführt werden. Deshalb ist es mit vertretbarem Aufwand nur möglich, eine grobe Näherung der optimalen Verteilung zu berechnen. Diese Alternative führt also zu einem schlechteren Ergebnis und belastet die WF-Server zusätzlich. Deshalb wird sie verworfen.

3.4.2 Entfernte Ausführung von Subprozessen

Eine Variante zur Partitionierung von WF ist die entfernte Ausführung von Subprozessen. Das heißt, das Granulat der Verteilung ist ein Sub-WF und nicht eine Aktivität. Verteilung auf Basis von Subprozessen wird z.B. von FlowMark ab Version 2.2 [IBM96a, IBM96b] und MOBILE [NSS98, Sch97b, SNS99] verwendet. Bei diesen Ansätzen wird nicht die WF-Instanz zu einem anderen WF-Server migriert, sondern es wird ein Sub-WF auf einem anderen Server gestartet. Der Vorteil dieser Vorgehensweise ist, dass nicht alle Instanzdaten migriert werden müssen. Stattdessen müssen nur die vom dem Sub-WF potentiell benötigten Daten übergeben werden. Dies führt prinzipiell zu einem kleineren zu übertragenden Datenvolumen. Werden aber die in Kapitel 6 vorgestellten Optimierungen bei Migrationen eingesetzt, so wird dieser Vorteil egalisiert. Das zu übertragende Datenvolumen ist dann höchstens so groß, wie bei einem entfernten Subprozess-Aufruf, da bei Migrationen nur die tatsächlich am Zielserver benötigten Daten übertragen werden. Das Datenvolumen kann sogar kleiner sein, wenn in einem Subprozess Daten nur manchmal benötigt werden (z.B. von Aktivitäten einer bedingten Verzweigung), die zwar beim entfernten Starten eines Subprozesses übertragen werden müssen, aber nicht bei den in Kapitel 6 vorgestellten Verfahren. Durch die bei Migrationen durchgeführten Optimierungen entfällt der Vorteil von entfernter Sub-WF-Ausführung also völlig.

Der entfernte Aufruf von Subprozessen hat zwei generelle Nachteile: Erstens muss nach Beendigung eines Subprozesses stets zum Server des Superprozesses zurückgekehrt werden. Warum dies ein Nachteil ist, soll an dem Beispiel aus Abb. 3.8 (Seite 56) erläutert werden: O.B.d.A. wird angenommen, dass der Super-WF in Rio de Janeiro kontrolliert wird. Dann muss nach Ausführung der Aktivität 20 in Denver zum Server in Rio de Janeiro zurückgekehrt werden, obwohl die Nachfolgeraktivität vom Server in Stuttgart kontrolliert wird. Dieser „Umweg bei der Kommunikation“ ist bei ADEPT nicht notwendig, da die WF-Instanz direkt von Denver nach Stuttgart migriert wird. Der zweite Nachteil betrifft die Flexibilität bei der Verteilung. Soll die Zuordnung von Aktivitäten zu WF-Servern geändert werden, so müssen die betroffenen Aktivitäten bei der diskutierten Alternative in einen anderen Sub-WF verschoben werden, da gesamte Subprozesse vom selben Server kontrolliert werden. Dies erfordert die Ummodellierung von Subprozessen, während in ADEPT nur ein Attribut der betroffenen Aktivität (die Serverzuordnung) verändert werden muss. Aus den genannten Gründen ist es günstiger, als Verteilungsgranularität Aktivitäten anstelle von Subprozessen zu verwenden.

[8] Eine Diskussion dieser Fragestellung für abhängige Bearbeiterzuordnungen und variable Serverzuordnungen findet sich im Abschnitt 5.1.1.2.

3.4.3 Physische Zerteilung der Prozessvorlage

In Mentor [WWWK96b, MWW+98] werden State- und Activitycharts physisch partitioniert. Die entstehenden Partitionen werden zur verteilten Steuerung der WF-Instanzen verwendet. Bei METEOR$_2$ [DKM+97] wird die Prozessbeschreibung zerteilt und kompiliert, so dass für jeden Aktivitätenknoten ein eigener Scheduler entsteht. Durch diese Vorgehensweisen sind den WF-Servern die vollständigen Vorlagen der von ihnen kontrollierten WF-Instanzen nicht bekannt.

Eine Alternative ist, die WF-Vorlagen bei allen WF-Servern repliziert zu speichern (zumindest bei denen, die evtl. eine Instanz dieses WF-Typs kontrollieren werden). Die Zuordnung der WF-Server zu den Aktivitäten erfolgt dann nicht durch eine physische Zerteilung der WF-Vorlage, sondern explizit durch Serverzuordnungsausdrücke. Auch diese können bei allen Servern repliziert gespeichert werden. Die Replikation verursacht kaum Kosten, da die Schemadaten sehr statisch sind. Diese Vorgehensweise ist günstiger, weil das Wissen über die vollständige Struktur einer WF-Instanz für zahlreiche Funktionalitäten eines WfMS benötigt wird. So erfordern z.B. variable Serverzuordnungen (siehe Kapitel 5), dass die Struktur gewisser Partitionen mehreren WF-Servern bekannt ist. Auch für die dynamische Änderung von WF-Instanzen zur deren Ausführungszeit [RD98, RBD99] ist es notwendig, die gesamte Struktur des WF zu kennen (siehe Kapitel 8). Aus diesen Gründen haben wir uns für diese Vorgehensweise entschieden.

3.5 Zusammenfassung

In diesem Kapitel wurde gezeigt, dass es essentiell ist, eine geeignete Systeminfrastruktur zu verwenden, um das beim verteilten WF-Management existierende Optimierungspotential voll auszuschöpfen. Deshalb wurde eine Systeminfrastruktur entworfen, bei der in jedem Teilnetz des Kommunikationsnetzwerks ein WF-Server angesiedelt ist. Außerdem wurde gezeigt, dass das Konzept der Partitionierung benötigt wird, um effizientes verteiltes WF-Management zu ermöglichen, d.h. eine WF-Instanz muss abschnittsweise von verschiedenen WF-Server kontrolliert werden können. Dies erfordert, dass die Kontrolle über eine WF-Instanz zu einem anderen WF-Server migriert werden kann. Um Migrationen bzgl. der Kommunikationskosten effizient realisieren zu können, wurden geeignete Verfahren zur Übertragung der Zustands- und Anwendungsdaten der WF-Instanz entwickelt. Außerdem wurde untersucht, wie diese Daten an Synchronisationspunkten wieder zusammengeführt werden können. Schließlich wurde noch ein Verfahren zum Zurücksetzen entwickelt, das bei der „normalen“ WF-Ausführung keinen zusätzlichen Aufwand generiert. Mit diesem Rüstzeug ist es möglich, eine effiziente verteilte WF-Ausführung zu realisieren. Allerdings wurde noch nicht diskutiert, wie für einen konkreten WF eine geeignete Verteilung der einzelnen Aktivitäten auf die WF-Server ermittelt werden kann. Dies erfolgt in den nun folgenden beiden Kapiteln.

Kapitel 4

Statische Serverzuordnungen

Wie wir in Kapitel 3 gesehen haben, ist es für das Kommunikationsverhalten eines WfMS günstig, den WF-Server einer Aktivität so zu wählen, dass er nahe bei deren potentiellen Bearbeitern liegt. Um dies zu ermöglichen, müssen die WF partitioniert werden, was den Einsatz von Migrationen notwendig macht. Doch auch diese belasten die WF-Server und verursachen Kommunikationskosten. Deshalb sollen sie in unserem Ansatz nur dann eingesetzt werden, wenn sie das Kommunikationsverhalten insgesamt verbessern. Das heißt, die Serverzuordnungen eines WF-Typs sind so zu wählen, dass der bei der Ausführung von Instanzen dieses Typs entstehende Kommunikationsaufwand minimiert wird. Da die entstehenden Kommunikationskosten schwer abzuschätzen sind, wäre der WF-Modellierer mit der Festlegung einer solchen Verteilung überfordert. Deshalb soll er dabei vom WfMS unterstützt werden: Der Modellierer definiert die WF-Typen wie im zentralen Fall und gibt zusätzlich einige statistischen Daten vor. Auf Basis dieser Information berechnet das WfMS die günstigsten Serverzuordnungen, die der Modellierer dann noch überarbeiten kann. Damit das WfMS die Qualität einer Verteilung beurteilen kann, benötigt es ein Kostenmodell, mit dem bei gegebenen Serverzuordnungen die entstehenden Kosten ermittelt werden können. Dieses wird von einem Algorithmus, der die optimale Verteilung berechnet, verwendet. Unseres Wissens ist ADEPT$_{distribution}$ der einzige Ansatz, bei dem eine Kostenrechnung durchgeführt wird, um die Qualität von Verteilungen zu beurteilen, und um auf diese Weise bzgl. den Kommunikationskosten optimale Serverzuordnungen zu berechnen. In diesem Kapitel wird wieder angenommen (wie in Kapitel 3), dass hauptsächlich unabhängige Bearbeiterzuordnungen verwendet werden, weshalb sich statische Serverzuordnungen ergeben. In Kapitel 5 wird der Sachverhalt für den Fall abhängiger Bearbeiterzuordnungen und variabler Serverzuordnungen diskutiert.

In Abschnitt 4.1 wird untersucht, welche Ideen aus dem Bereich der Scheduling-Verfahren für Betriebssystemprozesse auf unser Problem übertragbar sind. In Abschnitt 4.2 wird erläutert, welche Daten benötigt werden, um zur Modellierungszeit die bei der WF-Ausführung entstehenden Kosten abschätzen zu können. Diese Daten werden in Abschnitt 4.3 verwendet, um ein Kostenmodell aufzustellen und in Abschnitt 4.4, um die vom Kostenmodell benötigten Wahrscheinlichkeitsverteilungen (WV) zu berechnen. In Abschnitt 4.5 werden einige Algorithmen vorgestellt, die, unter Verwendung des Kostenmodells, eine geeignete Verteilung der Aktivitäten auf die WF-Server berechnen. Ein Algorithmus zum Ermitteln einer optimalen Verteilung der Benutzer auf die Teilnetze wird in Abschnitt 4.6 entwickelt. Schließlich wird in Abschnitt 4.7 noch nachgewiesen, dass das vorgestellte Verteilungsmodell effizient ist. Dabei wird insbesondere gezeigt, dass sich die insgesamt zu bewältigende Kommunikationslast durch Migrationen reduzieren lässt. Abschnitt 4.8 bietet eine Zusammenfassung dieses

Kapitels und einen Ausblick auf weitergehende Ideen.

4.1 Vorüberlegungen

Um ein Verfahren zur Zuordnung von Aktivitäteninstanzen zu WF-Servern zu entwickeln, wollen wir zuerst eine ähnliche, schon gut verstandene, Fragestellung betrachten: Bei der Lastbalancierung in Betriebssystem ist zu entscheiden, welchem Prozessor ein bestimmter Prozess zugeteilt werden soll. Zu beachten ist, dass die Betriebsystemprozesse bei unserer Fragestellung nicht den WF-Prozessen entsprechen. Eine WF-Instanz besteht aus Aktivitäteninstanzen, die den WF-Servern zugeteilt werden sollen, ebenso wie eine Anwendung aus Betriebssystemprozessen besteht, die den Prozessoren zugeteilt werden. Die Betriebssystemprozesse entsprechen also den Aktivitäteninstanzen. Um ein Verfahren für die Verteilung von Aktivitäten zu entwickeln, wollen wir nun Scheduling-Verfahren für Betriebssystemprozesse betrachten.

4.1.1 Dynamische Scheduling-Verfahren

Das Ziel von dynamischen Scheduling-Verfahren ist, die Last der Prozessoren so zu balancieren, dass sie alle ungefähr gleich stark ausgelastet sind. Sie bilden den linken Teilbaum in der Klassifikation [Cas81, CK88, Gos91] der globalen Lastbalancierungsalgorithmen[1] in Abb. 4.1.

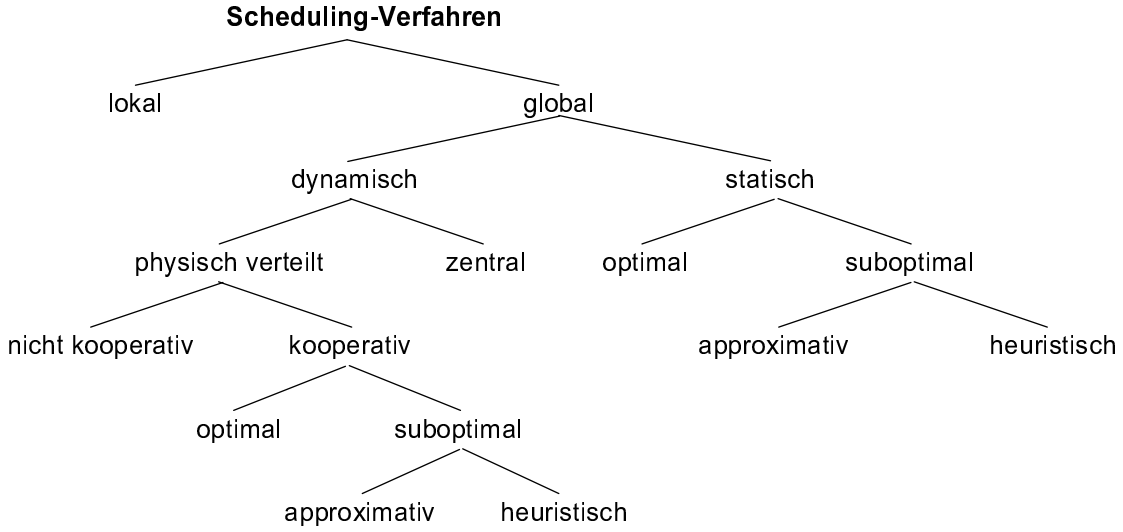


Abbildung 4.1 Klassifikation von Lastbalancierungsalgorithmen nach [CK88].

Dynamische Lastbalancierungsalgorithmen können physisch verteilt oder zentral ablaufen. Im nicht verteilten Fall, trifft ein zentraler Scheduler die Entscheidung, welcher Prozessor welchen Prozess ausführt. Nachteile dieser Variante sind, dass die Information über die aktuelle Auslastung aller Prozessoren zum zentralen Scheduler transportiert werden muss, und dass dieser einen Flaschenhals darstellt, was die Verfügbarkeit beeinträchtigt. Beim verteilten Scheduling wird die Entscheidung von mehreren Prozessoren gemeinsam getroffen, wobei die jeweils lokal vorhandene Information über die

[1] Als lokales Scheduling wird die Aufteilung der Zeitscheibe eines einzelnen Prozessors auf die lokalen Prozesse bezeichnet. Dies ist eine andere Fragestellung, als die hier untersuchte und entspricht eher der Zuordnung von Aktivitäteninstanzen zu „typgleichen“ Servern innerhalb desselben Teilnetzes. Dieser Aspekt wird in Kapitel 7 diskutiert.

Auslastung der Prozessoren genutzt wird. Verteilte Scheduling Algorithmen können kooperativ sein, d.h. die verschiedenen Komponenten verfolgen ein gemeinsames Ziel. Ist dies nicht der Fall, so wird nur die Performance des lokalen Prozessors optimiert. Dies kann dazu führen, dass die verschiedenen Prozessoren einander entgegengesetzt wirkende Entscheidungen treffen. Kooperative Scheduling-Algorithmen können die optimale oder eine suboptimale Lösung ermitteln. Eine suboptimale Lösung kommt bei Anwendung von approximativen Methoden oder von Heuristiken zustande.

Zusätzlich zu der soeben beschriebenen hierarchischen Klassifikation gibt es nach [CK88, Gos91] auch noch eine flache Klassifikation von Scheduling-Verfahren. In dieser werden weitere Eigenschaften der Verfahren betrachtet, wie z.B.:

- Wird für die Scheduling-Entscheidung nur der aktuelle Systemzustand betrachtet oder auch das ehemalige Verhalten. Im zweiten Fall erhält man einen adaptiven Scheduler, d.h. der Scheduling-Algorithmus wird im Laufe der Zeit verändert.
- Ergreift die Quelle (source-initiative) oder das Ziel (server-initiative) die Initiative. Im ersten Fall, sucht sich ein überlasteter Prozessor einen weniger belasteten, um einen Prozess an ihn abzutreten. Im zweiten Fall sucht sich ein unterlasteter Prozessor selbst Aufgaben.
- Der Grad an Information über den Zustand der Prozessoren, der zwischen ihnen geteilt wird. Zu diesem Zweck werden in [WM85] 7 Stufen des Informationsaustauschs definiert und die dadurch entstehenden Verfahren bewertet.
- Schließlich gibt es noch Verfahren, die ausschließlich beim Start eines Prozesses einen Prozessor zuordnen (non-pre-emptive) und welche, bei denen auch ein laufender Prozess zu einem anderen Prozessor migriert werden kann (pre-emptive). Die zweite Variante ist zwar machbar, aber mit sehr hohen Kosten verbunden, da ein laufender Prozess schon sehr viele Daten erzeugt haben kann, die ebenfalls migriert werden müssen.

In [Gos91] werden zu den einzelnen Kategorien konkrete Verfahren vorgestellt.

4.1.2 Statische Scheduling-Verfahren

In Betriebssystemen ist i.d.R. sehr wenig oder überhaupt keine Information über die Eigenschaften eines zu startenden Prozesses vorhanden. Deshalb werden üblicherweise dynamische Scheduling-Verfahren verwendet. In WfMS sind die Struktur der WF-Typen und die Eigenschaften der Aktivitäten (z.B. potentielle Bearbeiter) aber sehr wohl bekannt. Deshalb entspricht das Szenario eher demjenigen, welches bei statischen Scheduling-Verfahren vorausgesetzt wird: Der Ressourcenverbrauch der Betriebssystemprozesse ist vor deren Start bekannt. Damit lässt sich ein bzgl. der Charakteristika gut geeigneter Prozessor auswählen.

Bei statischen Scheduling-Verfahren (siehe Abb. 4.1) wird zwischen Varianten unterschieden, welche die optimale Verteilung der Prozesse auf die Prozessoren ermitteln, und solchen, welche nur eine suboptimale Lösung berechnen [Cas81, CK88, Gos91]. Diese kann aufgrund von Heuristiken oder durch Approximation ermittelt werden. Beim Berechnen der optimalen Lösung und bei der Approximation einer suboptimalen Lösung werden dieselben Verfahren angewandt, nämlich das (teilweise) Durchsuchen des Lösungsraums.

4.1.3 Diskussion der Verfahren

Um ein für den Einsatz in einen WfMS geeignetes Verfahren auswählen zu können, soll nun noch auf die Probleme der verschiedenen Verfahren eingegangen werden. Außerdem wird untersucht, inwie-

weit die bei den Scheduling-Verfahren gemachten Annahmen auch bei WfMS gelten.

In [ELZ86] werden verschiedene Scheduling-Verfahren analytisch verglichen. Dabei wird bei dynamischen Verfahren das Problem des „Processor Thrashing“ aufgezeigt, d.h. alle Knoten verwenden fast ihre komplette Zeit damit, Prozesse zu transferieren. Dieses Problem kann z.B. entstehen, wenn Schwellwerte falsch gewählt wurden. Da die Festlegung dieser Werte in einem WfMS durch die vergleichsweise komplexe Struktur der WF noch schwieriger ist, kann das Problem beim Einsatz dynamischer Verfahren in WfMS ebenfalls auftreten.

In [Sta84] werden Lastbalancierungsverfahren durch Simulationen verglichen. Daraus ergibt sich, dass die Kosten für den Scheduling-Algorithmus niedrig sein müssen, um ein gutes Ergebnis zu erzielen. Dies ist bei komplexen Lastbalancierungsverfahren nicht immer gegeben. [Sta84] betont außerdem, dass veraltete Lastinformation das Ergebnis eines dynamischen Scheduling-Verfahrens deutlich verschlechtern kann. Auch dies spricht gegen den Einsatz solcher Verfahren in einem WfMS, da diese häufig weiträumig verteilt sind, was den schnellen und häufigen Austausch von Lastinformation erschwert.

Die aufgeführten Aspekte sprechen gegen den Einsatz dynamischer Scheduling-Verfahren und für statische Verfahren. Bei diesen stimmen auch die Annahmen besser mit denen eines WfMS überein, da Information über die Aktivitäten vorhanden ist. Diese sollte auch genutzt werden. Dynamisches Scheduling führt zu einer gleichmäßigen Verteilung der Last auf die Prozessoren. Eine Reduzierung der insgesamt entstehenden Kommunikationslast lässt sich damit aber nicht erreichen, weil nicht geeignete Server gewählt werden, sondern nur aktuell (vergleichsweise) wenig belastete. Da sich deshalb eine wichtige Anforderung an effizientes WF-Management mit dynamischen Verfahren nicht erfüllen lässt, werden die statischen Verfahren weiter verfolgt. Im nachfolgenden Abschnitt wird beschrieben, wie die benötigte Information zu den WF-Typen beschafft werden kann. Anschließend werden Verfahren zum statischen Scheduling von WF-Aktivitäten entwickelt.

4.2 Statistische Daten

Um die bei einer gegebenen Verteilung (Serverzuordnungen) entstehenden Kosten abschätzen zu können, wird zusätzliche Information benötigt. So ist z.B. zur Abschätzung der durch die Aktivitätenausführung entstehenden Last relevant, wie häufig Aktivitäten eines bestimmten Typs gestartet werden und wie groß die dabei transferierte Datenmenge ist. Für solche Daten genügt jeweils ein Durchschnittswert, da die durch die Ausführung aller WF-Instanzen entstehende Gesamtlast minimiert werden soll. Es ist nicht primäres Ziel, die Ausführung einer einzelnen WF-Instanz zu optimieren.

4.2.1 Motivation

In MENTOR [WKM+95, WWWK96a, WWWK96b] und WIDE [CGP+96, CGS97] wird der WF-Server einer Aktivität immer in demjenigen Domain gewählt, in dem sich die potentiellen Bearbeiter dieser Aktivität befinden. Um eine größere Flexibilität zu ermöglichen, müssen in ADEPT nicht alle potentiellen Bearbeiter einer Aktivität demselben Domain angehören. Soll der WF-Server einer Aktivität so gewählt werden, dass er sich in demjenigen Domain befindet, welchem die meisten potentiellen Bearbeiter angehören, so wird außer den Bearbeiterzuordnungen die Lokation (zugehöriger Domain) der Bearbeiter benötigt. Diese Information ist dem WfMS bekannt.

In ADEPT$_{distribution}$ ist es aber nicht das Ziel, die Ausführung der einzelnen Aktivitäten getrennt zu optimieren. Stattdessen sollen die Ausführungen eines WF insgesamt optimiert werden. Deshalb müssen auch die Migrationskosten berücksichtigt werden. Dazu wird ein Kostenmodell benötigt, mit dem die bei einer gegebenen Verteilung insgesamt entstehenden Kosten berechnet werden können. Nur so lassen sich die durch eine Modifizierung der Verteilung entstehenden veränderten Kosten für die Aktivitätenausführung und für Migrationen miteinander vergleichen und gegeneinander aufrechnen. Dann kann entschieden werden, ob die Modifikation sinnvoll ist. Um dieses Kostenmodell erstellen zu können, wird zusätzliche Information über den WF benötigt. Diese ist teilweise schon im WfMS vorhanden, teilweise muss sie noch erfasst werden. Im Folgenden wird detailliert erläutert, um welche Daten es sich dabei handelt und wie diese ermittelt werden können.

4.2.2 Kosten einzelner Operationen

Um die Kosten für eine einzelne Aktivitätenausführung oder Migration berechnen zu können, wird Information über die involvierten Benutzer und über die transferierten Datenvolumina benötigt. Einige dieser Daten sind dem WfMS schon bekannt oder können aus ihm bekannter Information abgeleitet werden. In diesem Abschnitt wird untersucht, welche dies sind und welche Daten zusätzlich festgelegt werden müssen. Letztere müssen vom WF-Modellierer geschätzt werden. Wird der entsprechende WF schon von einem (evtl. zentralen) WfMS ausgeführt, so können die entsprechenden Daten auch durch Monitoring ermittelt werden.

4.2.2.1 Domain der potentiellen Bearbeiter

Bei der Ausführung einer Aktivität n muss der WF-Server mit Clients kommunizieren, um Arbeitslisten zu aktualisieren oder um Parameterdaten zu dem Aktivitätenprogramm zu transportieren (vgl. Abb. 3.1 auf Seite 49). Um die dabei entstehenden Kosten abschätzen zu können, muss die Menge der potentiellen Bearbeiter der Aktivität n bekannt sein ($PotBearb_n$). Um diese ermitteln zu können, werden die Bearbeiterzuordnungen der Aktivitäten benötigt (vgl. Abschnitt 2.4.3.2). Dies stellt aber keine zusätzliche Anforderung dar, da die Bearbeiterzuordnungen auch im zentralen Fall festgelegt werden müssen.

Um abschätzen zu können, in welchen Teilnetzen Kosten anfallen, muss außerdem noch bekannt sein, zu welchem Domain die einzelnen Bearbeiter $u \in PotBearb_n$ gehören. Deshalb muss der Domain, zu dem ein Benutzer gehört, dem WfMS bekannt gemacht werden. Häufig ist dieser schon bekannt, da in großen Anwendungen nicht jeder Benutzer an jedem Arbeitsplatz arbeiten kann, sondern seine Rechte auf seine eigene OE eingeschränkt sind. Deshalb können diese Domains vom WfMS häufig automatisch ermittelt werden.

4.2.2.2 Gewichtung der Bearbeiter

Es gibt Fälle, in denen ein Benutzer in verschiedenen Domains arbeiten darf (vgl. Abschnitt 2.4.1). Solche Fälle sollen bei der Berechnung der optimalen Serverzuordnungen berücksichtigt werden. Zu diesem Zweck wird für jeden Benutzer u ein Gewicht $G(u)$ eingeführt, das angibt, welchen Anteil der normalen Arbeitszeit dieser Benutzer aktiv ist. Ein Benutzer, der in mehreren Domains aktiv ist, kann bei der Kostenanalyse dann wie mehrere Benutzer behandelt werden, die jeweils nur einen Teil der Zeit aktiv sind. So kann ein Arzt z.B. mit einem Gewicht von 0,9 in seiner Station berücksichtigt

werden und mit 0,1 in der Notaufnahme. Die Gewichte können außerdem verwendet werden, um Benutzer zu modellieren, die Teilzeit arbeiten oder die nur einen Teil ihres Arbeitstags mit dem WfMS arbeiten. Die Gewichte der Benutzer müssen (zumindest, wenn sie vom Standardwert 1 abweichen) vom Modellierer vorgegeben werden. Sie werden von zahlreichen Algorithmen verwendet, um den entsprechenden Benutzer mehr oder weniger stark zu berücksichtigen.

Die Gewichte können auch von der aktuell betrachteten Aktivität n abhängen ($G_n(u)$). Die Gewichte eines Bearbeiters werden zwar häufig für alle Aktivitäten identisch sein, es gibt aber auch praktisch relevante Fälle, in denen die Gewichte für einzelne Aktivitäten vom Standardwert abweichen. So kann ein Benutzer „Lieblingsaktivitäten“ haben oder Aktivitäten, die er nur in Ausnahmefällen ausführt, wenn alle anderen potentiellen Bearbeiter gerade verhindert sind. Außerdem kann es Aktivitäten geben, die ein Benutzer nur in bestimmten Zeiträumen bearbeitet. So ist vielleicht ein Schalter nur vormittags geöffnet, so dass die entsprechenden Aktivitäten nur vormittags bearbeitet werden, während andere Tätigkeiten ganztägig ausgeführt werden. Für diese Aktivitäten ist $G_n(u)$ entsprechend größer bzw. kleiner.

Für die Gewichte eines Bearbeiters ist sogar eine noch feinere Unterscheidung möglich. Für einen Benutzer u können zwei Arten von benutzerspezifischen Gewichten $G_n(u)$ und $G'_n(u)$ unterschieden werden. Mit $G_n(u)$ legt der Modellierer fest, welchen Anteil der Zeit der Benutzer die *Aktivität n ausführt* (so dass durch den Parametertransfer Last erzeugt wird). Mit $G'_n(u)$ wird angegeben, welchen Anteil der üblichen Arbeitszeit er durchschnittlich am WfMS *angemeldet ist* (so dass seine Arbeitsliste aktualisiert werden muss). Ist die Aktivität n eine „Lieblingsaktivität“ des Benutzers u, so ist $G_n(u)$ größer als der Standardwert $G(u)$ dieses Benutzers. Auch das Gewicht $G'_n(u)$, das den Anteil an der Arbeitszeit beschreibt, in dem der Benutzer angemeldet ist, kann für verschiedene Aktivitäten unterschiedlich sein. Benutzer verwenden oft mehrere Arbeitslisten für unterschiedliche Gruppen von Aktivitäten. Wenn z.B. ein Benutzer für Aktivitäten am Schalter nur vormittags zuständig ist und deshalb die entsprechende Arbeitsliste nur vormittags geöffnet hat, so wird für diese Aktivitäten $G'_n(u) < G'(u)$ gewählt. Im Folgenden wird aber auf die Unterscheidung von $G_n(u)$ und $G'_n(u)$ verzichtet und das Gewicht eines Bearbeiters u als $G_n(u)$ bezeichnet. Falls die Unterscheidung jedoch benötigt wird, so hängt die Auswahl des bei einer Berechnung verwendeten Gewichtes davon ab, ob die Kosten für die Aktivitätenausführung ($G_n(u)$) oder für Arbeitslisten-Updates ($G'_n(u)$) berechnet werden sollen.

Durch die Gewichte wird beschrieben, dass die verschiedenen Benutzer eine zu bearbeitende Aktivität mit unterschiedlicher Wahrscheinlichkeit auswählen. Wenn die Gewichte der Benutzer bei einer Berechnung berücksichtigt werden, so kann zwischen den Benutzern Unabhängigkeit angenommen werden, d.h. die Wahrscheinlichkeit, dass ein Benutzer u die Aktivität n auswählt, ist so groß, wie sein Anteil an dem „Gesamtgewicht für Aktivität n“:
P(Benutzer u bearbeitet die Aktivität n) = $G_n(u) \;/ \sum_{u' \in PotBearb_n} G_n(u')$

4.2.2.3 Übertragene Datenmenge

Für die Abschätzung der Kosten einer Kommunikation ist auch das erwartete Datenvolumen relevant. Bei der Aktualisierung von Arbeitslisten hängt dies von dem verwendeten Verfahren (vgl. Abschnitt 2.5) und von der Größe der Einträge ab. Das durchschnittliche Datenvolumen beim Einfügen ($WL\_insert\_size_n$) und Löschen ($WL\_delete\_size_n$) eines Eintrags zur Aktivität n ist vom Modellierer zu schätzen (häufig wird die Größe für Einträge aller Aktivitätentypen identisch sein). Die Größe

des beim Start ($In\_parameter\_size_n$) bzw. Ende ($Out\_parameter\_size_n$) eines Aktivitätenprogramms für den Parametertransfer benötigten Datenvolumens hängt von den transferierten Datenelementen ab. Welche dies sind, ist dem WfMS bekannt, da der Datenfluss vollständig modelliert wird.[2] Die Größe eines Datenelements steht für skalare Typen (z.B. Integerwerte) fest. Bei Strings oder Multimediadaten ist häufig nur eine Obergrenze bekannt. Da für die zu erwartenden Kosten die durchschnittliche Größe der Datenelemente relevant ist, muss diese vom Modellierer geschätzt werden. Durch Addition der durchschnittlichen Größe der Datenelemente kann das System dann das beim Starten bzw. Beenden einer Aktivität transferierte Datenvolumen berechnen. Ebenso kann der für die Migration von Datenelementen (Abschnitt 3.3.4) erwartete Datenumfang berechnet werden. Wird dazu noch der Aufwand zur Migration der Zustandsinformation (Abschnitt 3.3.3) addiert, so erhält man das bei der Migration von Aktivität m zur Aktivität n erwartete Datenvolumen $Migration\_size_{m,n}$. Um die durch die Kommunikation des Anwendungsprogramms von Aktivität n mit einer externen Datenquelle in Teilnetz l entstehende Netzlast abschätzen zu können, müssen auch die dabei übertragenen Ein- ($Ext\_in_n(l)$) und Ausgabedatenmengen ($Ext\_out_n(l)$) bekannt sein. Da das WfMS kein Wissen über diese Kommunikationen hat, kann es auch nicht bei der Abschätzung der Datenvolumina assistieren. Diese müssen vom Modellierer auf Basis von Wissen über die Aktivitätenprogramme geschätzt werden.

4.2.3 Ausführungshäufigkeiten

Mit den im vorherigen Abschnitt festgelegten Daten ist es möglich, die Kosten für eine einzelne Ausführung einer Aktivität oder einer Migration abzuschätzen. Um die insgesamt entstehenden Kosten berechnen zu können, muss aber auch bekannt sein, wie häufig eine solche Aktion stattfindet. Wie diese Ausführungshäufigkeiten ermittelt werden können, wird in diesem Abschnitt untersucht.

Prinzipiell gibt es drei Möglichkeiten, um die Ausführungshäufigkeiten zu ermitteln:

1. Der Modellierer schätzt die Ausführungshäufigkeiten. Dabei kann er durch den am Ende dieses Abschnitts angegebenen Algorithmus 4.1 unterstützt werden.
2. Die Daten können durch Monitoring eines laufenden Systems gewonnen werden. Dazu muss der betreffende WF aber schon in einem realen System laufen. Bei diesem kann es sich aber auch um ein zentrales WfMS handeln, da lediglich die Häufigkeit der Aktivitätenausführung und des Signalisierens von Kanten gemessen werden muss.
3. Es ist auch möglich die Information durch eine Simulation zu ermitteln. Um diese durchführen zu können, werden aber dieselben Eingangsdaten benötigt, wie für Algorithmus 4.1. Da dieser die benötigten Daten analytisch ermittelt und da das Erstellen einer Simulationsumgebung einen beträchtlichen Aufwand erfordert, wird diese Variante nicht weiter verfolgt.

Es ist auch möglich die Varianten 1. und 2. zu kombinieren: Der Modellierer schätzt die Häufigkeiten und ermöglicht so die Berechnung von initialen Serverzuordnungen. Mit diesen wird die Arbeit im verteilten WfMS begonnen. Danach findet ständiges Monitoring statt. Mit den ermittelten Daten wird periodisch eine neue geeignete Verteilung berechnet und die Serverzuordnungen werden entsprechend angepasst. Durch diese Vorgehensweise ist es möglich, Schätzfehler zu korrigieren und auf Veränderungen im WfMS zu reagieren.

[2] Bei dokumentenorientierten WfMS (vgl. Abschnitt 2.1.1) wird der Datenfluss nicht vorgegeben, sondern es werden alle vorhandenen Dokumente zu jeder Aktivität transportiert. Da deren Gesamtgröße ebenfalls abgeschätzt werden kann, ist es auch bei diesen Systemen möglich, das bei der Aktivitätenausführung übertragene Datenvolumen zu berechnen.

Für den Modellierer wäre es sicher sehr schwierig, zu schätzen, wie häufig die einzelnen Aktivitäten ausgeführt und die Kanten signalisiert werden. Es ist wesentlich einfacher, die folgenden Werte anzugeben:

- Häufigkeit des Startens der WF-Instanz
- Wahrscheinlichkeit für die Wahl eines bestimmten Zweiges bei einer exklusiven Verzweigung
- Die durchschnittliche Anzahl von Iterationen einer Schleife

Aus diesen Daten lassen sich die Ausführungshäufigkeiten der Aktivitäten und Kanten berechnen. Allerdings ist dies bei Zugrundelegung eines beliebigen (z.B. Petri-Netz-basierten) WF-Modells keineswegs einfach, weil z.B. die Äste einer bedingten Verzweigung, die jeweils mit einer vorgegebenen Wahrscheinlichkeit gewählt werden, nicht in einem einzigen Punkt zusammengeführt werden müssen. Ebenso sind Schleifen möglich, die z.B. aus einem Ast einer bedingten Verzweigung heraus vor den Beginn dieser Verzweigung führen. Auf Grund solcher Schwierigkeiten ist die Berechnung der benötigten Ausführungshäufigkeiten i.Allg. sehr aufwendig. So wird in [GWWK99, GWWK00] ein sehr komplexes Modell vorgestellt, um eine ähnliche Fragestellung zu lösen. Dieses basiert das auf der Erstellung und Analyse von Continous-Time-Markov-Chains (CTMC). Prinzipiell ist es möglich, dieses Verfahren zu verwenden, um die vorliegende Problemstellung zu lösen.

Die klare Blockstrukturierung von ADEPT ermöglicht es, die Ausführungshäufigkeiten sehr viel einfacher zu berechnen. Deshalb wird im Folgenden ein Verfahren vorgestellt, das die Blockstrukturierung nutzt, um die benötigten Daten auf äußerst effiziente Art und Weise zu ermitteln. Dazu berechnet der Algorithmus 4.1 für jede Aktivität bzw. jede Kante die Wahrscheinlichkeit $p(n)$ bzw. $p(m,n)$, mit der sie ausgeführt wird (wobei Schleifen noch ignoriert werden). Außerdem wird die durch Schleifen begründete Anzahl der Ausführungen $\#It(n)$ bzw. $\#It(m,n)$ berechnet. Betrachten wir zuerst die Ausführungswahrscheinlichkeit der Aktivitäten $p(n)$: Die Startaktivität $Start$ einer WF-Instanz wird sicher ausgeführt ($p(Start) = 1$). Bei bedingten Aufspaltungen wird genau ein Zweig gewählt. Die Wahrscheinlichkeit dieses Zweiges sei p. Dann haben alle Aktivitäten dieses Zweiges eine um den Faktor p kleinere Ausführungswahrscheinlichkeit, als der Verzweigungsknoten. Betrachten wir nun die Anzahl der Iterationen $\#It(n)$: Die Startaktivität $Start$ wird einmal ausgeführt ($\#It(Start) = 1$). Aktivitäten einer Schleife mit durchschnittlich x Iterationen, werden x Mal so oft ausgeführt, wie die umgebenden Aktivitäten. Dies können bei verschachtelten Schleifen auch mehr als x Ausführungen sein. Sind die Werte $p(n)$ und $\#It(n)$ für alle Aktivitäten bekannt, so können sie auch für alle Kanten $m \rightarrow n$ berechnet werden. Dabei werden für Kanten vom Typ $CONTROL\_E$ und $LOOP\_E$ nur Signalisierungen mit TRUE_SIGNALED gezählt, bei Synchronisationskanten zusätzlich Signalisierungen mit FALSE_SIGNALED, da in genau diesen Fällen eine Migration stattfinden kann (vgl. Abschnitt 3.3.2). Aus den so berechneten Werten und der Ausführungshäufigkeit der WF-Instanz lassen sich die Häufigkeiten der Aktivitätenausführungen ($NodeFreq(n)$) und der Kantensignalisierungen ($EdgeFreq(m,n)$) berechnen.

Die Funktionsweise von Algorithmus 4.1 wird anhand des Beispiels aus Abb. 4.2 detailliert erläutert: Für die Startaktivität a ergibt sich $p(a) = 1$ und $\#It(a) = 1$. Normalerweise wird eine Aktivität mit der gleichen Wahrscheinlichkeit ausgeführt, wie ihre Vorgängeraktivität (z.B. $p(a) = 1 \rightarrow p(b) = 1$). Aktivitäten, die auf einen OR-Split-Knoten folgen, haben eine entsprechend kleinere Ausführungswahrscheinlichkeit ($p(m) = 0,6 \rightarrow p(n) = 0,18$, weil die Wahrscheinlichkeit für den Zweig von n 0,3 beträgt). Die Ausführungswahrscheinlichkeit des Join-Knotens ist so groß wie die des Split-Knotens ($p(m) = 0,6 \rightarrow p(p) = 0,6$). Da bei einer Parallelität stets alle Zweige ausgeführt werden, ist die Wahrscheinlichkeit der betroffenen Aktivitäten so groß wie die des AND-Split-Knotens ($p(i) = 0,6 \rightarrow p(j) = 0,6$). Auch der Anzahl der Iterationen einer Aktivität ist normalerweise so

groß, wie die der Vorgängeraktivität ($\#It(a) = 1 \rightarrow \#It(b) = 1$). Für Aktivitäten einer Schleife erhöht sich diese Anzahl ($\#It(c) = 5 \rightarrow \#It(d) = 50$, weil diese Schleife durchschnittlich 10 Mal durchlaufen wird). Für Aktivitäten, die auf den zugehörigen Schleifenendeknoten folgen, erniedrigt sich diese Anzahl entsprechend ($\#It(f) = 50 \rightarrow \#It(g) = 5$). Der Algorithmus ist so formuliert, dass er auch bei direkt aufeinander folgenden Schleifen funktioniert. Das Problem dabei ist, dass die Vorgängeraktivität (z.B. s) des Schleifenstartknotens (t) in dem Wert $\#It(s) = 3$ eine Schleife berücksichtigt, welche die Aktivität t überhaupt nicht betrifft. Deshalb darf nicht ohne weiteres von diesem Wert ausgegangen werden. Algorithmus 4.1 übernimmt zuerst diesen Wert $\#It(t) = \#It(s) = 3$. Dann wird erkannt, dass es sich bei t um den Startknoten einer Schleife handelt: $\#It(t) = iterations_t \cdot \#It(t) = 21$. Da es sich beim Vorgänger von t um den Schleifenendeknoten s handelt, wird mit $\#It(t) = 1/iterations_q \cdot \#It(t) = 7$ das richtige Ergebnis ermittelt.

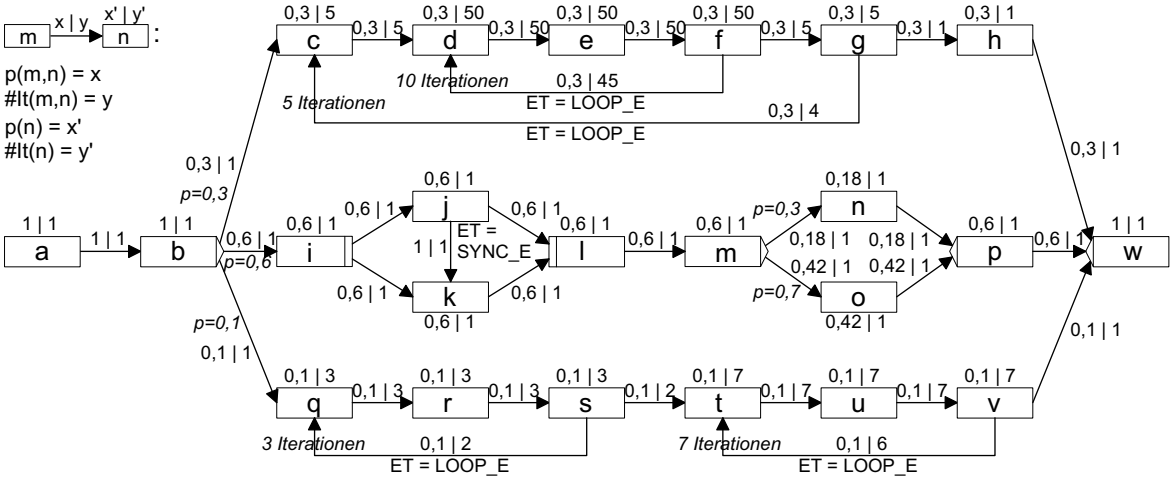


Abbildung 4.2 Beispiel für Häufigkeiten zu Algorithmus 4.1.

Bei Kontrollkanten $m \rightarrow n$ werden normalerweise die Werte der Vorgängeraktivität m übernommen. Bei der ersten Kante eines OR-Splits muss die Ausführungswahrscheinlichkeit allerdings noch mit der Wahrscheinlichkeit für den jeweiligen Zweig multipliziert werden ($p(m) = 0,6 \rightarrow p(m,n) = 0,18$). Eine Kontrollkante, mit der eine Schleife verlassen wird (z.B. $f \rightarrow g$), wird nur bei der letzten Schleifeniteration signalisiert, so dass ihre Anzahl von Iterationen niedriger ist, als die der Vorgängeraktivität ($\#It(f) = 50 \rightarrow \#It(f,g) = 5$). Bei den vorherigen Iterationen wird die Schleifenkante signalisiert ($\#It(f) = 50 \rightarrow \#It(f,d) = 45$). Synchronisationskanten sind zwar in genauso viele Iterationen involviert, wie ihre Quellaktivität ($\#It(j) = 1 \rightarrow \#It(j,k) = 1$), sie sind aber auch dann relevant, wenn der Zweig der Quellaktivität nicht gewählt wird. Da sie stets TRUE_SIGNALED oder FALSE_SIGNALED signalisiert werden und in beiden Fällen eine Migration notwendig wird (vgl. Abschnitt 3.3.2.2) beträgt ihre Ausführungswahrscheinlichkeit 1 ($p(j) = 0,6 \rightarrow p(j,k) = 1$).

4.3 Kostenmodell

In diesem Abschnitt wird ein Verfahren zur Berechnung der bei der WF-Ausführung auftretenden Kosten entwickelt. Dieses verwendet die im vorherigen Abschnitt vorgestellten statistischen Daten. Außerdem werden einige Wahrscheinlichkeitsverteilungen (WV) benötigt. Wie diese im Falle statischer Serverzuordnungen ermittelt werden, wird im Abschnitt 4.4 erläutert. Das Kostenmodell ist

Algorithmus 4.1 (Berechnung der Ausführungshäufigkeiten)
input
$prob_{m,n}$: für OR-Split-Knoten m die Wahrscheinlichkeit, dass der Zweig mit der Nachfolgeraktivität n gewählt wird
$iterations_n$: für Schleifenstartknoten n die durchschnittliche Anzahl von Durchläufen der Schleife
$instance\_frequency$: Häufigkeit von WF-Instanzen des untersuchten Typs
$CFS = (N, E, NT, V_{in}, V_{out}, \ldots)$: Kontrollflussstruktur des untersuchten WF-Typs
(wird implizit auch von Funktionen wie $Pred_{Typ}(n)$ verwendet)
output
$NodeFreq(n)$: $\forall n \in N$: Ausführungshäufigkeit von Aktivitäteninstanzen des Typs n
$EdgeFreq(m,n)$: $\forall (n,m,*) \in E$: Häufigkeit der (True-) Signalisierung von Kanten dieses Typs
begin
// Analyse für alle Aktivitäten n
for each Aktivität $n \in N$ (in partieller Ordnung bezüglich Kontrollfluss) **do**
if $NT(n) =$ STARTFLOW **then** // n ist die Startaktivität des WF
$p(n) = 1$; $\#It(n) = 1$;
else
wähle beliebiges $m \in Pred_{\{CONTROL\_E\}}(n)$;
// $p(n)$ für Aktivität n berechnen
$p(n) = p(m)$;
if $V_{out}(m) =$ *ONE_Of_ALL* **then** // m ist ein OR-Split-Knoten
$p(n) = prob_{m,n} \cdot p(m)$;
if $V_{in}(n) =$ *ONE_Of_ALL* **then** // n ist ein OR-Join-Knoten
$p(n) = p(Split_n)$;
// $\#It(n)$ für Aktivität n berechnen
$\#It(n) = \#It(m)$;
if $NT(n) =$ *STARTLOOP* **then** // n ist Startknoten einer Schleife
$\#It(n) = iterations_n \cdot \#It(n)$;
if $NT(m) =$ *ENDLOOP* **then** // n ist Nachfolger eines Schleifenendeknotens
$\#It(n) = 1/iterations_{Startloop(m)} \cdot \#It(n)$;

// Analyse für alle Kanten e
for each $e = (m, n, edgeType) \in E$ **do**
case $edgeType = CONTROL\_E$: // e ist eine Kontrollkante
$p(m,n) = p(m)$; $\#It(m,n) = \#It(m)$; // Daten der Quellaktivität von e übernehmen
if $V_{out}(m) =$ *ONE_Of_ALL* **then** // e ist erste Kante eines OR-Splits
$p(m,n) = prob_{m,n} \cdot p(m)$;
if $NT(m) =$ *ENDLOOP* **then** // e ist Kante nach einem Schleifenendeknoten
$\#It(m,n) = 1/iterations_{Startloop(m)} \cdot \#It(m)$;
case $edgeType = LOOP\_E$: // e ist eine Schleifenkante
$p(m,n) = p(m)$;
$\#It(m,n) = (iterations_n - 1)/iterations_n \cdot \#It(m)$;
case $edgeType = SYNC\_E$: // e ist eine Synchronisationskante
$p(m,n) = 1$; $\#It(m,n) = \#It(m)$;

// Berechnung der Ausführungshäufigkeiten
for each $n \in N$ **do**
$NodeFreq(n) = instance\_frequency \cdot p(n) \cdot \#It(n)$;
for each $e = (m, n, *) \in E$ **do**
$EdgeFreq(m,n) = instance\_frequency \cdot p(m,n) \cdot \#It(m,n)$;
end.

allgemeiner gehalten, als es für statische Serverzuordnungen notwendig wäre. Dadurch kann es unverändert für die Kostenberechnung bei variablen Serverzuordnungen (siehe Kapitel 5) übernommen werden. Bei diesen ist aber die Berechnung der WV wesentlich komplizierter.

Ein Kostenmodell zur Berechnung der Kosten bzw. zur Bestimmung der Last für die unter Performanz-Aspekten kritischen Komponenten des WfMS (Server, Teilnetze, Gateways) muss zumindest die folgenden Kosten berücksichtigen:

- Kosten für den Transfer von Ein- und Ausgabeparameterdaten beim Starten und Beenden von Aktivitätenprogrammen
- Kosten für das Aktualisieren von Arbeitslisten
- Kosten für die Migration von WF-Instanzen
- Kosten für die Kommunikation von Aktivitätenprogrammen mit externen Datenquellen

Im Kostenmodell werden einige WV verwendet (siehe auch Anhang A): Im Folgenden bezeichnet $ExProb_n(i,j)$ die Wahrscheinlichkeit, dass die Aktivität n vom Server in Teilnetz i kontrolliert und von einem Benutzer in Teilnetz j bearbeitet wird. $MigrProb_{m,n}(i,j)$ sei die Wahrscheinlichkeit, dass beim Übergang von Aktivität m nach n vom Server i zum Server j migriert werden muss. $\#User_n(j|i)$ sei die Anzahl der potentiellen Bearbeiter von Aktivität n im Domain j, wenn diese Aktivität vom Server im Teilnetz i kontrolliert wird. Diese WV hängen von den gewählten Serverzuordnungen ab. Wie die WV bestimmt werden können, wird für den Fall statischer Serverzuordnungen in Abschnitt 4.4 gezeigt. Für den Moment wollen wir sie als gegeben betrachten. Im Folgenden werden Formeln entwickelt, die das anfallende bzw. zu transportierende Datenvolumen für jede Aktion (Aktivität ausführen, Migration, ...) beschreiben, und zwar getrennt für jede Komponente des WfMS. Diese Formeln werden anschließend in eine Gesamtformel eingesetzt, mit der die in einer Komponente entstehende Last berechnet werden kann.

4.3.1 Aktivitätenausführung

Der Erwartungswert für das Datenvolumen (im Folgenden kurz: Datenvolumen) bei der Aktivitätenausführung (Transport der Ein- und Ausgabeparameterdaten) berechnet sich wie folgt: Das Datenvolumen, das am Server i für die Ausführung von Aktivität n entsteht, ergibt sich als Wahrscheinlichkeit, dass Server i die Aktivität n kontrolliert, multipliziert mit der zu transportierenden Datenmenge:

$$Vol^{Akt}_{Server,i}(n) = \Big(\sum_j ExProb_n(i,j)\Big) \cdot (In\_parameter\_size_n + Out\_parameter\_size_n)$$

Die Belastung für die Teilnetze ergibt sich folgendermaßen: Im Teilnetz i findet Kommunikation statt, wenn entweder der Server in i liegt oder wenn ein Bearbeiter aus Teilnetz i gewählt wird. Trifft beides zu, so wird die Kommunikation nur einmal gezählt (deshalb $j \neq i$):

$$\begin{aligned} Vol^{Akt}_{TN,i}(n) = &\Big(\sum_j ExProb_n(i,j) + \sum_{j\neq i} ExProb_n(j,i)\Big) \\ &\cdot(In\_parameter\_size_n + Out\_parameter\_size_n) \end{aligned}$$

Das am Gateway von Teilnetz i nach j anfallende Datenvolumen ($i \neq j$) ergibt sich wie folgt:

$$Vol^{Akt}_{GW,i,j}(n) = ExProb_n(i,j) \cdot In\_parameter\_size_n + ExProb_n(j,i) \cdot Out\_parameter\_size_n$$

4.3.2 Aktualisieren der Arbeitslisten

Die Berechnung der Datenvolumina für das Aktualisieren der Arbeitslisten ist etwas problematisch. Dies liegt u.a. daran, dass für das Aktualisieren verschiedene Verfahren verwendet werden können (siehe Abschnitt 2.5), aus denen unterschiedliche Kosten resultieren. Bei der Abschätzung der Datenvolumina muss außerdem berücksichtigt werden, dass es relevant ist, ob bei jeder Aktualisierung die gesamte Arbeitsliste oder nur neu hinzugekommene oder entfernte Einträge übertragen werden.

Im Folgenden wird das Verfahren analysiert, bei dem jede Änderung sofort an alle betroffenen Clients propagiert wird. Für die anderen Verfahren sind ähnliche Abschätzungen denkbar. Allerdings hängen die Kosten dann nicht nur von der WF-Ausführung ab, sondern z.B. auch davon, ob die Frequenz der neuen Einträge deren Zusammenfassung erlaubt. Aufgrund solcher Aspekte wird eine Abschätzung aufwendig und ungenau. Eine Möglichkeit dem zu begegnen ist, die Kosten vom Modellierer manuell abschätzen oder überarbeiten zu lassen. Allerdings lohnt sich dieser Aufwand kaum, da Arbeitslisten-Einträge relativ klein sind. Die durch ihren Transfer entstehenden Datenvolumina sind im Vergleich zum Parametertransfer und den Migrationen fast vernachlässigbar. Diese Aussage wurde auch durch Simulationen (siehe Kapitel 9) bestätigt. Deshalb ist die im Folgenden beschriebene Berechnungsmethode völlig ausreichend. Wird ein optimiertes Verfahren zum Aktualisieren der Arbeitslisten verwendet, so wird durch die beschriebene Abschätzung eine obere Schranke für die entstehenden Datenvolumina ermittelt. Dies gilt, weil wir das aufwendigste Verfahren analysieren, bei dem jede Änderung sofort an alle betroffenen Clients propagiert wird. Um realistischere Ergebnisse zu erhalten, können bei optimierten Verfahren die berechneten Werte noch mit einem vom Modellierer vorzugebenen Faktor F ($0 \leq F \leq 1$) multipliziert werden. Wenn die Kosten für Arbeitslisten-Aktualisierungen im Vergleich zu den anderen Kosten besonders klein sind, kann es sogar sinnvoll sein, sie völlig zu ignorieren ($F = 0$).

Der WF-Server i muss Arbeitslisten aktualisieren, wenn die Aktivität n durch ihn kontrolliert wird; unabhängig davon in welchem Teilnetz j der spätere Bearbeiter angesiedelt ist. Um die Anzahl der Benutzer, deren Arbeitslisten aktualisiert werden müssen, zu erhalten, werden die potentiellen Bearbeiter aller Teilnetze k zusammengezählt. Das Datenvolumen, das bei einem einzigen Einfügen und späteren Löschen eines Arbeitslisteneintrags für Aktivität n übertragen wird, ist $WL\_insert\_size_n$ und $WL\_delete\_size_n$. Dies kann die Größe eines Eintrags oder der gesamten Arbeitsliste sein, je nachdem ob stets die gesamte Arbeitsliste oder nur die Änderungen übertragen werden. Das Datenvolumen für den WF-Server ergibt sich damit als:

$$Vol^{AL}_{Server,i}(n) = \sum_j ExProb_n(i,j) \cdot \sum_k \#User_n(k|i) \cdot (WL\_insert\_size_n + WL\_delete\_size_n)$$

Im Teilnetz i findet Kommunikation statt, wenn Server i Arbeitslisten aktualisiert oder wenn ein beliebiger anderer Server l die Arbeitsliste eines Bearbeiters aus Teilnetz i aktualisiert:

$$Vol^{AL}_{TN,i}(n) = \Big(\sum_j ExProb_n(i,j) \cdot \sum_k \#User_n(k|i) + \sum_{l \neq i} \cdot \sum_j ExProb_n(l,j) \cdot \#User_n(i|l)\Big) \cdot (WL\_insert\_size_n + WL\_delete\_size_n)$$

Das Gateway von Teilnetz i nach j muss Daten transferieren, wenn der Server des Teilnetzes i die Arbeitsliste eines Benutzers aus Teilnetz j aktualisiert. Auch hier ist nur die Wahrscheinlichkeit relevant, mit welcher der Server i die Aktivität kontrolliert, unabhängig davon in welchem Teilnetz l sie später bearbeitet wird:

$$Vol^{AL}_{GW,i,j}(n) = \sum_l ExProb_n(i,l) \cdot \#User_n(j|i) \cdot (WL\_insert\_size_n + WL\_delete\_size_n)$$

4.3.3 Kommunikation mit externen Datenquellen

Aktivitätenprogramme, die auf dem Rechner des Clients laufen, können mit externen Datenquellen (z.B. Datenbanken) kommunizieren. Dem WF-Server entsteht durch solche Kommunikationen kein Aufwand, da die Daten direkt zwischen Teilschrittprogramm und Datenquelle fließen:

$Vol^{Ext}_{Server,i}(n) = 0$

Im Teilnetz i findet Kommunikation zu einer Datenquelle im Teilnetz l statt, wenn der Client im TN i liegt (unabhängig vom Ort k des Servers). Außerdem wird das Teilnetz i belastet, wenn die Datenquelle von Aktivität n im Teilnetz i liegt und der Client in irgendeinem anderen Teilnetz l:

$$Vol^{Ext}_{TN,i}(n) = \sum_k ExProb_n(k,i) \cdot \sum_l (Ext\_in_n(l) + Ext\_out_n(l)) \\ + \sum_k \cdot \sum_{l \neq i} ExProb_n(k,l) \cdot (Ext\_in_n(i) + Ext\_out_n(i))$$

Vom Teilnetz i nach j fließen Eingabedaten, wenn sich die Datenquelle in i und der Client in j befindet, und Ausgabedaten, wenn der Client im Teilnetz i und die Datenquelle im Teilnetz j liegt:

$$Vol^{Ext}_{GW,i,j}(n) = \sum_k ExProb_n(k,j) \cdot Ext\_in_n(i) + \sum_k ExProb_n(k,i) \cdot Ext\_out_n(j)$$

Die Kosten, die durch Kommunikation der Anwendungsprogramme mit externen Datenquellen entstehen, sind unabhängig von der gewählten Serverzuordnung, da die Server in diesen Datentransfer nicht involviert sind. Sie hängen aber von der Wahl der Domains für die Benutzer und für die Datenquellen ab. Diese Festlegung kann mit Hilfe der Kostenabschätzung optimiert werden. Auch wenn der Modellierer dabei meist nur wenig Entscheidungsfreiheit hat, müssen die durch die gewählte Verteilung entstehenden Kosten bei der Analyse berücksichtigt werden, um die im Netzwerk entstehende Last abschätzen und so eine drohende Überlastung erkennen zu können.

4.3.4 Migrationen

Von besonderem Interesse sind in dem betrachteten Kontext natürlich die Migrationskosten beim Übergang von Aktivität m nach n. Hierbei müssen sowohl ein- wie auch ausgehende Migrationen berücksichtigt werden. Dabei ist zu beachten, dass keine Kommunikation stattfindet, wenn die Server für Aktivität m und n identisch sind (deshalb: $j \neq i$). Um die für Server i erwartete Datenmenge zu berechnen, wird das bei dieser Migration zu transportierende Datenvolumen mit der Wahrscheinlichkeit, dass diese Migration den Server i betrifft, multipliziert:

$$Vol^{Migr}_{Server,i}(m,n) = \sum_{j \neq i} \Big(MigrProb_{m,n}(i,j) + MigrProb_{m,n}(j,i) \Big) \cdot Migration\_size_{m,n}$$

Das durch die Migration verursachte Datenvolumen in den betroffenen Teilnetzen ist genau so groß wie für die Server:

$Vol^{Migr}_{TN,i}(m,n) = Vol^{Migr}_{Server,i}(m,n)$

Die Gateways müssen hierbei die folgenden Datenmengen transportieren:

$Vol^{Migr}_{GW,i,j}(m,n) = MigrProb_{m,n}(i,j) \cdot Migration\_size_{m,n}$

4.3.5 Gesamtkosten

Für jede Systemkomponente müssen diese Datenmengen noch mit den Ausführungshäufigkeiten der einzelnen Aktivitäten $NodeFreq(n)$ bzw. der Kanten $EdgeFreq(m,n)$ multipliziert und für alle

Aktivitäten aller WF-Typen (WF_Types) aufaddiert werden, um die entsprechende Last dieser Komponente zu bestimmen. Die dabei berechneten Werte können verwendet werden, um zu überprüfen, ob die entsprechende Komponente überlastet sein wird. Für die Server ergibt sich die Last wie folgt ($Load_{TN,i}$ und $Load_{GW,i,j}$ analog):

$$\begin{aligned} Load_{Server,i} = \sum_{\substack{CFS \in WF\_Types \\ \text{mit } CFS=(N,\ldots)}} \sum_{n \in N} \Big(& NodeFreq(n) \cdot Vol^{Akt}_{Server,i}(n) \\ & + NodeFreq(n) \cdot Vol^{AL}_{Server,i}(n) \\ & + NodeFreq(n) \cdot Vol^{Ext}_{Server,i}(n) \\ & + \sum_{\substack{m \in N \\ \text{mit } m \neq n}} EdgeFreq(m,n) \cdot Vol^{Migr}_{Server,i}(m,n) \Big) \end{aligned}$$

Bei unterschiedlichen Aktionen eines WfMS auftretende Verzögerungen werden von den Benutzern verschieden stark wahrgenommen. So bemerkt ein Benutzer nicht, wenn eine Migration lange dauert, da er bei dieser Aktion nicht auf eine Antwort des Systems wartet. Anders verhält es sich, wenn er eine Aktivitäteninstanz aus seiner Arbeitsliste ausgewählt hat und auf den Start des entsprechenden Aktivitätenprogramms wartet. Da dafür die Eingabeparameter vom WF-Server zu seinem Rechner transportiert werden müssen, wird hier eine langsame (weil weit entfernte) Verbindung zum WF-Server als besonders unangenehm empfunden. Eine Möglichkeit diesen Sachverhalt zu berücksichtigen, ist, die vorherige Formel wie nachfolgend angegeben abzuwandeln und die Kosten für den Transfer von Eingabedaten von Aktivitätenprogrammen mit einem Faktor $G_{Akt} > 1$ zu gewichten ($G_{AL} = G_{Ext} = G_{Migr} = 1$). Dadurch wird erreicht, dass z.B. Migrationskosten vergleichsweise weniger ins Gewicht fallen, weshalb die WF-Server tendenziell näher bei den Bearbeitern der Aktivitäten gewählt werden. Dadurch verringern sich die bewusst wahrgenommenen Wartezeiten. Die „Last“ für den WF-Server i ($Load'_{TN,i}$ und $Load'_{GW,i,j}$ analog) ergibt sich damit als:

$$\begin{aligned} Load'_{Server,i} = \sum_{\substack{CFS \in WF\_Types \\ \text{mit } CFS=(N,\ldots)}} \sum_{n \in N} \Big(& G_{Akt} \cdot NodeFreq(n) \cdot Vol^{Akt}_{Server,i}(n) \\ & + G_{AL} \cdot NodeFreq(n) \cdot Vol^{AL}_{Server,i}(n) \\ & + G_{Ext} \cdot NodeFreq(n) \cdot Vol^{Ext}_{Server,i}(n) \\ & + G_{Migr} \cdot \sum_{\substack{m \in N \\ \text{mit } m \neq n}} EdgeFreq(m,n) \cdot Vol^{Migr}_{Server,i}(m,n) \Big) \end{aligned}$$

Durch die zusätzlichen Gewichte entsprechen $Load'_{Server,i}$, $Load'_{TN,i}$ und $Load'_{GW,i,j}$ nicht der zu erwartenden Belastung der Komponenten. Deshalb kann mit diesen Werten nicht überprüft werden, ob sie überlastet sind. Stattdessen werden die Werte in eine zu minimierende Zielfunktion K eingesetzt, so dass die Gewichte Einfluss darauf haben, wie stark die einzelnen Aktionen gewichtet werden. Diese Zielfunktion wird berechnet, indem man die Last aller Komponenten mit komponentenspezifischen Kostenfaktoren $G_{Server,i}$, $G_{TN,i}$ und $G_{GW,i,j}$ gewichtet und aufaddiert. Diese Gewichte geben an, wie teuer es ist, ein Byte über den Server, das Teilnetz bzw. das Gateway zu transportieren. Der Modellierer kann damit beeinflussen, wie stark jede Komponente belastet werden soll. Durch einen großen Wert wird erreicht, dass z.B. eine bestimmte WAN-Verbindung (Gateway) wenig verwendet wird. Die zu minimierende Zielfunktion ergibt sich damit als:

$$K = \sum_i G_{Server,i} \cdot Load'_{Server,i} + \sum_i G_{TN,i} \cdot Load'_{TN,i} + \sum_i \sum_{j \neq i} G_{GW,i,j} \cdot Load'_{GW,i,j}$$

Die Wahl der Gewichte hängt davon ab, welches Ziel bei der Berechnung der Verteilung verfolgt wird. Die Gesamtbelastung der WF-Server kann durch eine geeignete Verteilung nicht minimiert werden, da jede Aktion von genau einem WF-Server erledigt werden muss. Die Gesamtlast ist umso kleiner, je

weniger Migrationen stattfinden. Das Ziel des vorgestellten Ansatzes ist es aber, das Kommunikationsverhalten zu optimieren. So ist es naheliegend $\forall i : G_{Server,i} = 0$ zu wählen. Angenommen, man will ausschließlich die durchschnittliche Netzlast minimieren, weil die Teilnetze stark belastet sind und die Gateways (z.B. in einem LAN) keinen Flaschenhals darstellen. Dann wählt man $\forall i : G_{TN,i} = 1$ und $\forall i, j : G_{GW,i,j} = 0$. Für einzelne besonders leistungsschwache Teilnetze kann auch $G_{TN,i} > 1$ festgelegt werden. Wenn hingegen bestimmte WAN-Verbindungen die Leistungsfähigkeit des Systems limitieren, so sollte für die zugehörigen Gateways $G_{GW,i,j} \gg 1$ gewählt werden.

Die durch die Ausführung von Instanzen unterschiedlicher WF-Typen entstehenden Gesamtkosten werden minimiert, indem die Kosten für die einzelnen WF-Typen getrennt minimiert werden. Dies ist möglich, weil sich die Gesamtlast einer Systemkomponente durch die Addition der Werte der von den einzelnen WF-Typen erzeugten Last ergibt. Die Last, die durch Instanzen eines bestimmten WF-Typs für eine Komponente resultiert, ist unabhängig von anderen WF-Typen, da verschiedene WF-Instanzen unabhängig voneinander ausgeführt werden. Wird nun jeder der (voneinander unabhängigen) Lastwerte minimiert, so führt dies zu der minimal möglichen Summe. Deshalb ist es legitim, mit den in Abschnitt 4.5 präsentierten Verteilungsalgorithmen, die optimalen Serverzuordnungen für jeden WF-Typen getrennt zu ermitteln. Lediglich wenn die Gesamtlast einer Systemkomponente (bei gegebenen Serverzuordnungen) berechnet werden soll (z.B. um zu überprüfen, ob diese überlastet sein wird), müssen alle WF-Typen gemeinsam betrachtet werden.

4.4 Berechnung der Wahrscheinlichkeitsverteilungen

In dem im vorherigen Abschnitt aufgestellten Kostenmodell werden die WV $ExProb_n(i,j)$, $MigrProb_{m,n}(i,j)$ und $\#User_n(j|i)$ verwendet. Diese WV hängen u.a. von den konkret gewählten Bearbeiter- und Serverzuordnungen ab. In diesem Abschnitt wird beschrieben, wie die WV bei ausschließlicher Verwendung statischer Serverzuordnungen berechnet werden können. Die Berechnung bei Verwendung variabler Serverzuordnungen ist wesentlich komplizierter. Sie wird in Kapitel 5 ausführlich beschrieben.

4.4.1 Berechnung von $ExProb_n(i,j)$

Die WV $ExProb_n(i,j)$ gibt an, mit welcher Wahrscheinlichkeit Aktivität n vom Server des Teilnetzes i kontrolliert und von einem Benutzer aus Teilnetz j bearbeitet wird. Im statischen Fall lautet die Serverzuordnung der Aktivität n: $ServZuordn_n = \mathtt{k}$, d.h. der WF-Server befindet sich bei Ausführung von Aktivität n stets im Teilnetz k. Für $k \neq i$ ist die oben angegebene Bedingung also nicht erfüllbar, so dass sich die Wahrscheinlichkeit 0 ergibt.

Im zweiten Teil der Bedingung wird gefordert, dass sich der Bearbeiter der Aktivität n im Teilnetz j befindet. Die Wahrscheinlichkeit, dass die zutrifft, entspricht dem Anteil der potentiellen Bearbeiter von Aktivität n, die im Teilnetz j angesiedelt sind, an der Gesamtheit dieser Bearbeiter. Bei der Berechnung dieses Anteils müssen die in Abschnitt 4.2.2.2 eingeführten Gewichte der Bearbeiter berücksichtigt werden. Damit ergibt sich:

$$ExProb_n(i,j) = \begin{cases} 0 & \text{, falls } ServZuordn_n \neq \mathtt{i} \\ \sum\limits_{\substack{u \in PotBearb_n \wedge \\ Domain(u)=j}} G_n(u) \;/ \sum\limits_{u \in PotBearb_n} G_n(u) & \text{, sonst} \end{cases}$$

4.4.2 Berechnung von $MigrProb_{m,n}(i,j)$

$MigrProb_{m,n}(i,j)$ ist die Wahrscheinlichkeit, dass beim Übergang von Aktivität m nach n vom Server i zum Server j migriert werden muss. Die Berechnung ist bei statischen Serverzuordnungen trivial, da die Serverzuordnung direkt den WF-Server der Aktivität angibt. Damit ergibt sich die Wahrscheinlichkeit 1 genau dann, wenn Aktivität m dem Server im Domain i und Aktivität n dem Server im Domain j zugeordnet ist:

$$MigrProb_{m,n}(i,j) = \begin{cases} 1 & \text{, falls } ServZuordn_m = \texttt{i} \wedge ServZuordn_n = \texttt{j} \\ 0 & \text{, sonst} \end{cases}$$

4.4.3 Berechnung von $\#User_n(j \mid i)$

$\#User_n(j|i)$ beschreibt die „Anzahl“ der potentiellen Bearbeiter von Aktivität n im Domain j, wenn diese Aktivität vom Server im Teilnetz i kontrolliert wird. Bei der Berechnung dieses Wertes müssen die Gewichte $G_n(u)$ der Benutzer u bei der Ausführung von Aktivität n berücksichtigt werden. $\#User_n(j|i)$ wird als 0 definiert, falls die Aktivität n nicht vom WF-Server im Domain i kontrolliert wird. Andernfalls müssen zur Berechnung von $\#User_n(j|i)$ die Gewichte der Benutzer des Domains j addiert werden:

$$\#User_n(j|i) = \begin{cases} 0 & \text{, falls } ServZuordn_n \neq \texttt{i} \\ \sum\limits_{\substack{u \in PotBearb_n \wedge \\ Domain(u)=j}} G_n(u) & \text{, sonst} \end{cases}$$

4.5 Ermitteln geeigneter Serverzuordnungen

Durch Verwendung des Kostenmodells aus Abschnitt 4.3 und der WV aus Abschnitt 4.4 ist es möglich, Verteilungen zu bewerten und zu vergleichen. Mit diesem Rüstzeug können Algorithmen entwickelt werden, die eine geeignete Verteilung der Aktivitäten auf die WF-Server (Serverzuordnungen) berechnen. Dabei muss zwischen der Performanz des verwendeten Verteilungsalgorithmus und der Güte der berechneten Verteilung abgewägt werden. Allerdings werden die Serverzuordnungen der Aktivitäten schon zur Modellierungszeit berechnet, d.h. die Ausführung der Verteilungsalgorithmen belastet die WF-Server nicht. Deshalb ist deren Performanz kein besonders kritischer Faktor. Es stellt sich lediglich die Frage, ob in vernünftiger Zeit überhaupt ein akzeptables Ergebnis gefunden wird. In dem nun folgenden Abschnitt werden verschiedene Verteilungsalgorithmen vorgestellt. Anschließend werden noch verwandte Algorithmen diskutiert und es wird aufgezeigt, welche Ideen übernommen wurden und wie weitere Varianten der Verteilungsalgorithmen entwickelt werden können.

Alle Algorithmen, die im Folgenden für statische Serverzuordnungen vorgestellt werden, können auch für die Berechnung variabler Serverzuordnungen verwendet werden. Dies ist möglich, weil die Algorithmen als Eingabeparameter für alle Aktivitäten $n \in N$ die Menge der potentiellen Serverzuordnungen $PotServZuordn_n$ haben. Angenommen, dass das WfMS aus den WF-Servern $s_1 \ldots s_{\#Serv}$ besteht, dann ergibt sich bei statischen Serverzuordnungen $\forall n : PotServZuordn_n = \{s_1, \ldots s_{\#Serv}\}$. In Fall variabler Serverzuordnungen werden diese Mengen entsprechend erweitert, d.h. für die Aktivitäten sind mehr Serverzuordnungen möglich als im statischen Fall.

Es gibt Situationen, in denen der Modellierer für einige Aktivitäten bestimmte Serverzuordnungen vorgeben oder verbieten will. Dies ist notwendig, wenn eine Aktivität von einem bestimmten WF-

Server kontrolliert werden muss (z.B. aus technischen Gründen oder aufgrund von Vorgaben bei unternehmensübergreifenden WF). Bei den beschriebenen Verteilungsalgorithmen ist es leicht möglich, für bestimmte Aktivitäten die Menge der erlaubten Serverzuordnungen einzuschränken. Dazu werden einer Verteilung, in der eine nicht erlaubte Serverzuordnung enthalten ist, die Kosten ∞ zugeordnet (Funktion $calculate\_costs$). Eine solche Verteilung wird von den Algorithmen niemals ausgewählt. Da auch Migrationen, in welche mit solchen Vorgaben behaftete Aktivitäten involviert sind, bei der Berechnung der Kosten berücksichtigt werden, haben diese Vorgaben Auswirkungen auf die Festlegung der Server benachbarter Aktivitäten. Es wird also die insgesamt optimale Verteilung unter Berücksichtigung der Vorgaben berechnet, was nicht der Fall wäre, wenn mit Vorgaben behaftete Aktivitäten einfach ignoriert werden würden. Der Modellierer hat natürlich die Möglichkeit, die berechnete Verteilung noch manuell zu ändern. Dabei kann er vom WfMS unterstützt werden, indem es ihm die vor und nach der Änderung entstehenden Kosten berechnet, so dass er die geplante Änderung bewerten kann.

4.5.1 Berechnung der optimalen Lösung

Um die optimalen Serverzuordnungen zu berechnen, können alle Kombinationen aller möglichen Serverzuordnungen $PotServZuordn_n$ aller Aktivitäten $n \in N$ erzeugt werden. Für jede dieser Kombinationen berechnet die Funktion $calculate\_costs(TestServZuordn_*, all)$[3], unter Verwendung des in Abschnitt 4.3 vorgestellten Kostenmodells, die entstehenden Kosten bei des Ausführung einer WF-Instanz dieses Typs. Schließlich wählt der Algorithmus 4.2 diejenige Serverzuordnung aus, die zu den minimalen Kosten führt.

Algorithmus 4.2 (Berechnung der optimalen Serverzuordnungen)
input
$CFS = (N, E, \ldots)$: Kontrollflussstruktur des WF-Typs mit $N = \{n_1, n_2, \ldots n_{|N|}\}$
$PotServZuordn_n$: $\forall n \in N$: die möglichen Serverzuordnungen für Aktivität n
output
$ServZuordn_n$: $\forall n \in N$: die optimale Serverzuordnung für Aktivität n
begin
$MinCost = \infty$;
for each $TestServZuordn_{n_1} \in PotServZuordn_{n_1}$ **do**
for each $TestServZuordn_{n_2} \in PotServZuordn_{n_2}$ **do**
...
for each $TestServZuordn_{n_{|N|}} \in PotServZuordn_{n_{|N|}}$ **do**
$Cost = calculate\_costs(TestServZuordn_*, all)$; [3]
if $Cost < MinCost$ **then**
$ServZuordn_* = TestServZuordn_*$;
$MinCost = Cost$;
end.

Da für $|N|$ Aktivitäten jeweils $\#Serv$ Serverzuordnungen getestet werden, hat der Algorithmus 4.2 die exponentielle Komplexität $O(\#Serv^{|N|})$. Ein WF kann durchaus $|N| > 100$ Aktivitäten umfassen. Dies kann dazu führen, dass es nicht möglich ist, in akzeptabler Zeit die optimalen Serverzuord-

[3] In der Funktion $calculate\_costs$ steht $TestServZuordn_*$ für die Serverzuordnungen aller Aktivitäten der WF-Vorlage. Lautet der zweite Parameter all, so werden die Kosten für die Ausführung aller Aktivitäten und aller Migrationen berechnet. Wird an dieser Stelle eine Aktivität n angegeben, so werden nur die Kosten zur Ausführung der Aktivität n (ohne Migrationskosten) berechnet.

nungen zu berechnen. Deshalb werden in den nachfolgenden Abschnitten Verfahren vorgestellt, die eine gute Näherung der optimalen Lösung berechnen. Danach wird in den Abschnitten 4.5.4.1 und 4.5.4.2 noch beschrieben, wie effizientere Verfahren zur Berechnung der optimalen Serverzuordnungen entwickelt werden können. Da auch diese ein komplexes Optimierungsproblem lösen müssen, benötigen sie dafür i.d.R. aber auch exponentiell viel Zeit.

4.5.2 Berechnung einer suboptimalen Lösung

Um zu vermeiden, dass der Verteilungsalgorithmus exponentielle Laufzeit benötigt, wird in diesem Abschnitt ein Algorithmus zur Berechnung der Serverzuordnungen vorgestellt, der auf einem Greedy-Ansatz [BB93, CLR96, Sch97a] basiert. Dieser Algorithmus 4.3 läuft in 2 Phasen ab: Zuerst wird jede Aktivität einzeln betrachtet und die für sie optimale Serverzuordnung wird berechnet. In der 2. Phase werden Partitionen zusammengefasst, falls die Migration zwischen ihnen nicht rentabel ist. Durch diese Vorgehensweise werden die folgenden Fälle unnötiger Migrationen vermieden, die ansonsten in der Praxis häufig vorkommen würden:

- Zwischen Aktivitäten, die vom selben Server kontrolliert werden, liegen wenige Aktivitäten, welche einem anderen WF-Server zugeordnet sind. Die Migrationen, zuerst zu diesem Server und danach wieder zurück, sind nicht rentabel. In diesem Fall ist es günstiger, alle Aktivitäten demselben Server (dem der äußeren Partitionen) zuzuordnen, um die Kosten für die Migrationen einzusparen.
- Zwischen zwei Partitionen befinden sich wenige Aktivitäten einer dritten Partition (oder evtl. sogar mehrerer Partitionen). Wenn sich wegen diesen wenigen Aktivitäten die Migration zu dem weiteren Server nicht lohnt, so ist es besser, diese Aktivitäten einer der äußeren Partitionen zuzuordnen.

Um Migrationskosten einzusparen müssen also gewisse Partitionen aufgelöst werden. Damit nicht alle möglichen Zusammenfassungen gleichzeitig untersucht werden müssen (was wieder zu einer exponentiellen Laufzeit führen würde), wird die folgende Heuristik verwendet, um die Kandidaten für eine Änderung der Serverzuordnung zu identifizieren: „Das Ändern der Serverzuordnungen einer Partition verursacht nur geringe Mehrkosten bei der Steuerung der einzelnen Aktivitäten, wenn die Partition klein ist, d.h. wenige Aktivitäten enthält.“ Deshalb prüft der Algorithmus 4.3 zuerst für kleine Partitionen, ob es rentabel ist, sie mit einer benachbarten Partition zusammenzufassen, so dass die Migration zwischen ihnen entfällt. Wurden alle kleineren Partitionen untersucht, so wird dies für größere Partitionen durchgeführt. Ein Beispiel für die Funktionsweise des Algorithmus 4.3 ist in Abb. 4.3 dargestellt.

In der Phase 1 von Algorithmus 4.3 wird zunächst eine Ausgangslösung bestimmt. Diese berücksichtigt noch keine Migrationskosten. Für jede Aktivität – einzeln betrachtet – werden alle möglichen Serverzuordnungen ausprobiert. Aus den potentiell möglichen Serverzuordnungen wird diejenige ausgewählt, die zu den minimalen Kosten für die Ausführung dieser Aktivität führt. Da die Aktivitäten dabei isoliert betrachtet werden, ergeben sich i.d.R. auch nicht rentable Migrationen. In der Phase 2 wird überprüft, welche dieser Migrationen sinnvoll sind. Dazu wird ausgerechnet, ob es günstiger ist, eine Partition P mit einer direkten Vorgänger- oder Nachfolgerpartition zusammenzufassen, so dass die Migration dazwischen entfällt. Wie schon erwähnt, betrachtet der Algorithmus zuerst kleine Partitionen. Diese werden mit Nachbarpartitionen zusammengefasst, wodurch immer größere Gruppen von Aktivitäten gebildet werden, zwischen denen keinen Migrationen stattfinden. Dies wird solange durchgeführt, bis keine unrentablen Migrationen mehr vorhanden sind. Dieses Vorgehen reduziert die

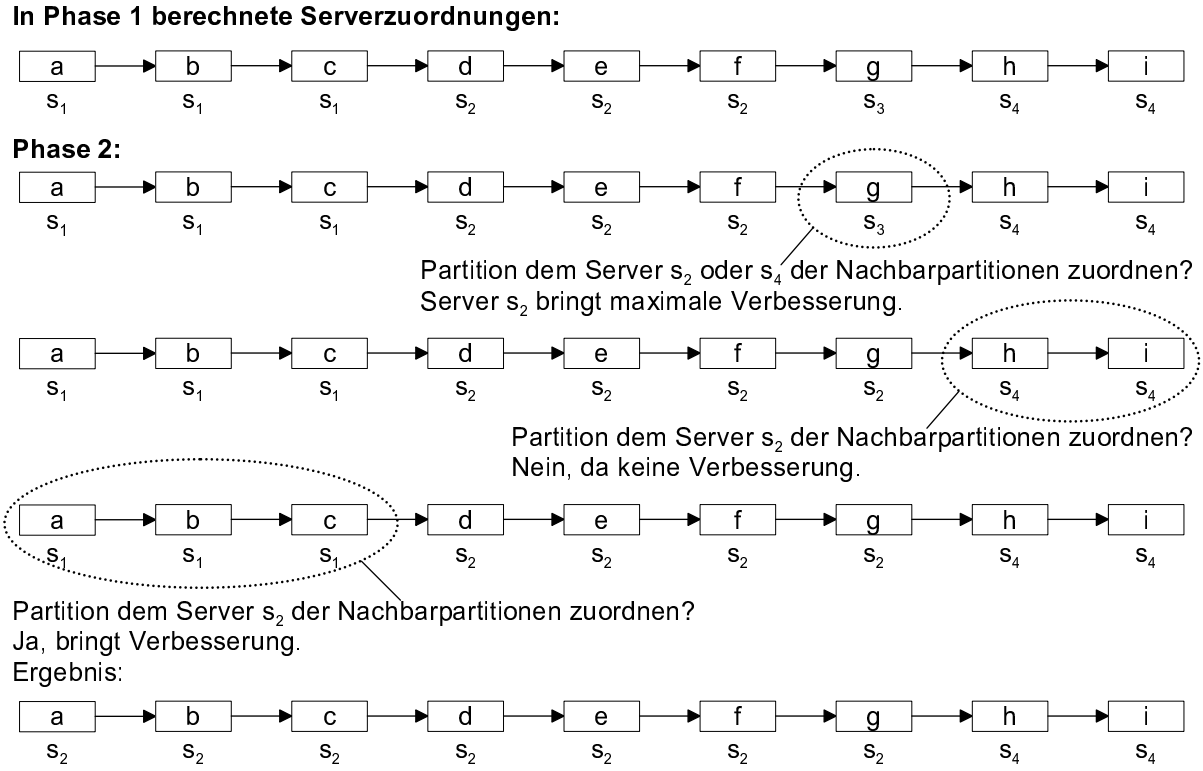


Abbildung 4.3 Beispiel für die Funktionsweise von Algorithmus 4.3.

zu kommunizierende Datenmenge bei der Ausführung der WF-Instanzen, da Partitionen genau dann zusammengefasst werden, wenn dies die Kommunikationskosten reduziert.

4.5.3 Verbesserungen des Verfahrens

Mit dem Algorithmus 4.3 können sehr gute Serverzuordnungen berechnet werden. Allerdings lassen sich Fälle konstruieren, in denen Partitionen nicht optimal zusammengefasst werden. Diese Problemfälle werden im Folgenden vorgestellt und es wird aufgezeigt, wie Algorithmus 4.3 modifiziert werden muss, um diese Fälle geeignet zu behandeln. Die dabei entstehenden Algorithmen weisen weiterhin ein polynomielles Laufzeitverhalten auf.

Auch die modifizierten Algorithmen berechnen nicht immer die optimale Verteilung, sondern nur eine sehr gute Näherung. Ein Grund dafür ist, dass die Algorithmen weiterhin versuchen, Paare von Partitionen zusammenzufassen. Es wird nicht versucht, für eine beliebige Teilmenge aller Partitionen den optimalen Server zu ermitteln. Da es bei $|N|$ Aktivitäten $O(|N|)$ Partitionen geben kann, können $O(2^{|N|})$ solche Teilmengen gebildet werden. Wenn sie alle analysiert werden, so hat der entstehende Algorithmus kein polynomielles Laufzeitverhalten mehr. Eine exponentielle Komplexität kann bei sehr großen WF-Vorlagen aber nicht akzeptiert werden. Deshalb wird der Ansatz, beliebige Mengen von Partitionen zu untersuchen, im Folgenden nicht verfolgt.

Algorithmus 4.3 (Berechnung gut geeigneter Serverzuordnungen)
input
$CFS = (N, E, \ldots)$: Kontrollflussstruktur des WF-Typs mit $N = \{n_1, n_2, \ldots n_{|N|}\}$
$PotServZuordn_n$: $\forall n \in N$: die möglichen Serverzuordnungen für Aktivität n
output
$ServZuordn_n$: $\forall n \in N$: eine günstige Serverzuordnung für Aktivität n
begin
Phase 1:
for each Aktivität $n \in N$ (in partieller Ordnung bezüglich Kontrollfluss) **do**
$MinCost = \infty$;
for each $TestServZuordn_n \in PotServZuordn_n$ **do**
$Cost = calculate\_costs(TestServZuordn_*, n)$;
if $Cost < MinCost$ **then**
$ServZuordn_n = TestServZuordn_n$;
$MinCost = Cost$;
Phase 2:
$MinCost = calculate\_costs(ServZuordn_*, all)$;
for *PartGröße* = 1 **to** $|N|$ **do**
for each $P : |P| =$*PartGröße*
$\wedge$ P ist max. Teilgraph mit $\forall l_1, l_2 \in P : ServZuordn_{l_1} \equiv ServZuordn_{l_2}$ **do**
$OptAct = NULL$;
for each $a \notin P$ mit $\exists l \in P$:
$a \in Pred_{\{CONTROL\_E, SYNC\_E, LOOP\_E\}}(l) \vee$
$a \in Succ_{\{CONTROL\_E, SYNC\_E, LOOP\_E\}}(l)$ **do**
for each $l \in P$ **do** $TestServZuordn_l = ServZuordn_a$;
for each $l \notin P$ **do** $TestServZuordn_l = ServZuordn_l$;
$TestCost = calculate\_costs(TestServZuordn_*, all)$;
if $TestCost < MinCost$ **then**
$OptAct = a$;
$MinCost = TestCost$;
if $OptAct \neq NULL$ **then**
for each $l \in P$ **do**
$ServZuordn_l = ServZuordn_{OptAct}$;
end.

Problemfall 1

Der erste Problemfall ergibt sich durch die Annahme, dass es günstig ist, zuerst kleine Partitionen mit Nachbarpartitionen zusammenzufassen. In Abb. 4.4 ist ein Beispiel dargestellt, in dem die daraus resultierende Heuristik versagt. Dabei ist die Partition des Servers s_2 kleiner als die des Servers s_1. Deshalb wird zuerst getestet, ob es eine Verbesserung bringt, wenn diese Partition vom Server s_1 kontrolliert wird. Da dies der Fall ist, gibt es keine Aktivitäten mehr, die vom Server s_2 kontrolliert werden, so dass nicht mehr getestet wird, ob es günstiger ist, wenn alle Aktivitäten vom Server s_2 kontrolliert werden. Da die Partition des Servers s_1 größer ist, ist es zwar unwahrscheinlich, aber durchaus möglich, dass es günstiger gewesen wäre, diese Aktivitäten dem Server s_2 zuzuordnen.

Der Algorithmus 4.4 löst das soeben beschriebene Problem. Die Heuristik wird nicht verwendet. Stattdessen werden alle möglichen Zusammenfassungen analysiert und diejenige ausgewählt, die zu der maximalen Verbesserung führt. Dadurch kann es nicht mehr vorkommen, dass die Aktivitäten einer kleineren Partition P_1 dem Server einer größeren Partition P_2 zugeordnet werden, obwohl der umge-

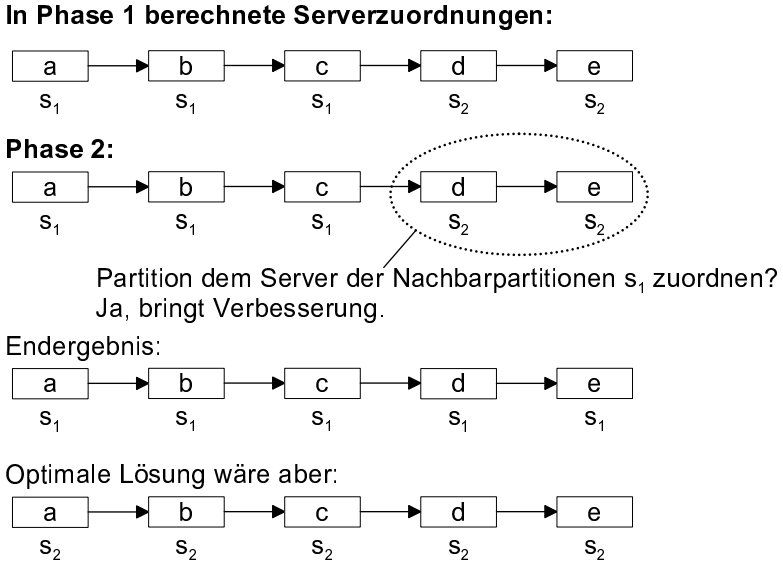


Abbildung 4.4 Beispiel in dem die von Algorithmus 4.3 verwendete Heuristik ungünstig ist.

Algorithmus 4.4 (Lösung für den Problemfall 1)
input
$CFS = (N, E, \ldots)$: Kontrollflussstruktur des WF-Typs mit $N = \{n_1, n_2, \ldots n_{|N|}\}$
$PotServZuordn_n$: $\forall n \in N$: die möglichen Serverzuordnungen für Aktivität n
output
$ServZuordn_n$: $\forall n \in N$: eine günstige Serverzuordnung für Aktivität n
begin
Phase 1:
// wie in Algorithmus 4.3
Phase 2:
$MinCost = calculate\_costs(ServZuordn_*, all)$;
repeat
$OptAct = NULL$;
for each $P : P$ ist max. Teilgraph mit $\forall l_1, l_2 \in P : ServZuordn_{l_1} \equiv ServZuordn_{l_2}$ **do**
for each $a \notin P$ mit $\exists\, l \in P$:
$a \in Pred_{\{CONTROL\_E, SYNC\_E, LOOP\_E\}}(l) \vee$
$a \in Succ_{\{CONTROL\_E, SYNC\_E, LOOP\_E\}}(l)$ **do**
for each $l \in P$ **do** $TestServZuordn_l = ServZuordn_a$;
for each $l \notin P$ **do** $TestServZuordn_l = ServZuordn_l$;
$TestCost = calculate\_costs(TestServZuordn_*, all)$;
if $TestCost < MinCost$ **then**
$OptAct = a$;
$OptPart = P$;
$MinCost = TestCost$;
if $OptAct \neq NULL$ **then**
for each $l \in OptPart$ **do**
$ServZuordn_l = ServZuordn_{OptAct}$;
until $OptAct = NULL$; // es wurde keine Verbesserung mehr gefunden
end.

kehrte Fall günstiger wäre. Allerdings erfordert dieser Algorithmus einen größeren Aufwand, da bei jedem Durchlauf der Repeat-Until-Schleife die Kosten aller möglichen Zusammenfassungen analysiert werden müssen. Die bei diesem Algorithmus gewählte Vorgehensweise, stets den Weg mit der maximalen Verbesserung zu wählen, wird in [Tur96, CLR96] z.B. auch bei Greedy-Algorithmen für das Traveling-Salesman-Problem vorgeschlagen.

Problemfall 2

Der zweite Problemfall entsteht, weil beim Zusammenfassen von zwei Partitionen nur die beiden Server berücksichtigt werden, die ursprünglich für die beteiligten Partitionen vorgesehen waren. Es kann aber vorkommen, dass für die entstehende Partition ein dritter Server optimal ist. Ein solches Beispiel ist in Abb. 4.5 dargestellt. Für die Aktivitäten a und b ist der Server s_1 optimal, der Server s_3 ist aber auch recht gut geeignet. Ähnlich verhält es sich für die Aktivitäten c und d: Der Server s_2 ist optimal und s_3 wiederum gut geeignet. Der Algorithmus 4.3 erzeugt nun eine der dargestellten Verteilungen, bei denen die Aktivitäten von den Servern s_1 bzw. s_2 kontrolliert werden. Die optimale Lösung wäre aber, alle Aktivitäten dem Server s_3 zuzuordnen. Dann ist keine Migration notwendig und alle Aktivitäten werden von einem „recht gut geeigneten“ Server gesteuert.

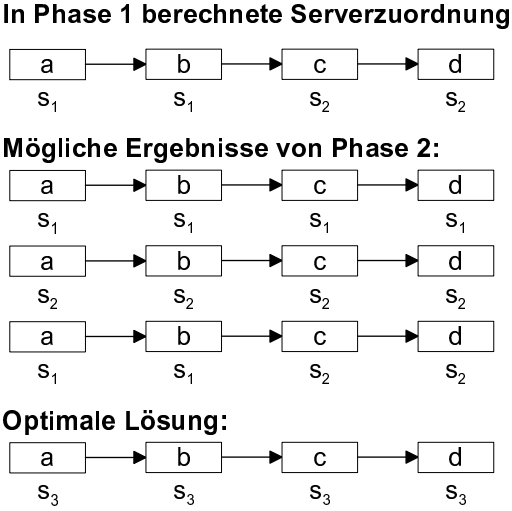


Abbildung 4.5 Beispiel bei dem der Algorithmus 4.3 nicht optimal Zusammenfassen kann.

Der Algorithmus 4.5 findet in dem in Abb. 4.5 dargestellten Fall die optimale Lösung. In der Phase 1 wird für jede Aktivität nicht nur die optimale Serverzuordnung ermittelt, sondern die K am besten geeigneten Serverzuordnungen ($ServZuordn_n[1..K]$). K ist eine vorzugebende Konstante. Ist K groß, so wird das Ergebnis besser, es wächst aber auch die Laufzeit des Algorithmus. Der Algorithmus 4.3 prüft beim Zusammenfassen einer Partition P mit einer Nachbarpartition P' (der Aktivität a) nur, ob die Verwendung der Serverzuordnung von P' zu den minimalen Kosten führt. Bei dem erweiterten Algorithmus 4.5 werden jeweils die K besten Serverzuordnungen aller Aktivitäten b der Nachbarpartition P' in Erwägung gezogen. Dadurch wird erreicht, dass ein Server, der sowohl für die Nachbarpartition P', wie auch für die untersuchte Partition P gut geeignet ist, für die gemeinsame Partition verwendet wird.

Der Algorithmus 4.5 kann auch in Verbindung mit Algorithmus 4.4 verwendet werden. Dann wird beim Zusammenfassen von zwei Partitionen nicht nur $ServZuordn_a$ als neue Serverzuordnung in

Algorithmus 4.5 (Lösung für den Problemfall 2)
input
$CFS = (N, E, \ldots)$: Kontrollflussstruktur des WF-Typs mit $N = \{n_1, n_2, \ldots n_{|N|}\}$
$PotServZuordn_n$: $\forall n \in N$: die möglichen Serverzuordnungen für Aktivität n
K: Anzahl der beim Zusammenfassen zu betrachtenden Serverzuordnungen
output
$ServZuordn_n$: $\forall n \in N$: eine günstige Serverzuordnung für Aktivität n
begin
Phase 1:
// wie in Algorithmus 4.3
// berechne zusätzlich $\forall n \in N$ die K günstigsten Serverzuordnungen: $ServZuordn_n[1..K]$
Phase 2:
$MinCost = calculate\_costs(ServZuordn_*, all)$;
for *PartGröße* = 1 **to** $|N|$ **do**
for each $P : |P| =$*PartGröße*
$\wedge$ P ist max. Teilgraph mit $\forall l_1, l_2 \in P : ServZuordn_{l_1} \equiv ServZuordn_{l_2}$ **do**
$OptAct = NULL$;
for each $a \notin P$ mit $\exists l \in P$:
$a \in Pred_{\{CONTROL\_E, SYNC\_E, LOOP\_E\}}(l) \vee$
$a \in Succ_{\{CONTROL\_E, SYNC\_E, LOOP\_E\}}(l)$ **do**
$P' =$ max. Teilgraph mit $a \in P'$ // die Partition der Aktivität a
$\wedge \forall l_1, l_2 \in P' : ServZuordn_{l_1} \equiv ServZuordn_{l_2}$;
for each $b \in P'$ **do**
for $i = 1$ **to** K **do**
for each $l \in P$ **do** $TestServZuordn_l = ServZuordn_b[i]$;
for each $l \notin P$ **do** $TestServZuordn_l = ServZuordn_l$;
$TestCost = calculate\_costs(TestServZuordn_*, all)$;
if $TestCost < MinCost$ **then**
$OptAct = b$;
$OptNr = i$;
$MinCost = TestCost$;
if $OptAct \neq NULL$ **then**
for each $l \in P$ **do**
$ServZuordn_l = ServZuordn_{OptAct}[OptNr]$;
end.

Erwägung gezogen, sondern es werden (wie in Algorithmus 4.5) für alle Aktivitäten b der Partition P' alle Serverzuordnungen $ServZuordn_b[i]$ für $i = 1 \ldots K$ betrachtet.

4.5.4 Verwandte Algorithmen

In diesem Abschnitt wollen wir einige Algorithmen betrachten, die in der Literatur zur Lösung von Optimierungsproblemen vorgeschlagen werden. Sie werden darauf hin untersucht, ob mit ihnen evtl. geeignete Serverzuordnungen ermittelt werden können. Wir betrachten sowohl Algorithmen zum Ermitteln des absoluten Optimums (Backtracking, Branch-and-bound) als auch zur Berechnung von Näherungen dieses Optimums (Hill-Climbing, Simulated-Annealing). Dabei wird aufgezeigt, inwieweit die entsprechenden Ideen in die Algorithmen 4.2 - 4.5 eingeflossen sind und warum bestimmte Ansätze für das vorliegende Problem weniger geeignet sind.

4.5.4.1 Backtracking

Durch Backtracking [BB93, GB65] wird stets die optimale Lösung für ein Problem berechnet. Beim Backtracking wird eine Teillösung versuchsweise erweitert und die Erweiterungen werden auch wieder zurückgenommen. Prinzipiell können dadurch alle Lösungsmöglichkeiten untersucht werden, jedoch werden Zweige, die nicht mehr zum Erfolg führen können, nicht weiter verfolgt. Dadurch wird der Aufwand reduziert, da i.d.R. nicht der gesamte Lösungsraum durchsucht werden muss.

Backtracking dient zum Durchsuchen von gerichteten, zyklenfreien Graphen (meist Bäumen). Der Graph ist dabei implizit, d.h. er existiert nicht physisch, sondern die Knoten lassen sich nur eindeutig beschreiben. Das Verfahren beginnt bei der Wurzel, welche die leere Lösung repräsentiert. Dann wird ein beliebiger Zweig gewählt, das heißt der Lösung wird ein Element hinzugefügt. Dies wird solange wiederholt, wie die Teillösung noch zu einer Gesamtlösung führen kann. Falls dies nicht mehr möglich ist, wird das letzte gewählte Element entfernt und durch eine alternative Lösungsmöglichkeit (Bruderzweig) ersetzt. Immer wenn eine Gesamtlösung entdeckt wird, wird diese ausgegeben. Bei Optimierungsproblemen kann eine Teillösung noch zu einer (optimalen) Gesamtlösung führen, wenn die durch die Teillösung verursachten Kosten kleiner sind, als die der besten bisher gefundenen Gesamtlösung. Die beste bisher gefundene Gesamtlösung muss gespeichert werden und kann am Ende des Algorithmus ausgegeben werden. Da ausschließlich Zweige ignoriert werden, die nicht mehr zu einer optimalen Lösung führen können, findet Backtracking stets das globale Optimum. Eine typische Anwendung für Backtracking ist das 8-Damen-Problem [BB93], bei dem 8 Damen so auf einem Schachbrett zu platzieren sind, dass sie sich nicht bedrohen.

Um die optimalen Serverzuordnungen mittels Backtracking zu berechnen, muss das Problem als Graph dargestellt werden (siehe Abb. 4.6). Der erste Knoten stellt die leere Lösung dar (alle Serverzuordnungen sind undefiniert). Seine Nachfolgerknoten sind die möglichen Serverzuordnungen für die Aktivität 1. Deren Nachfolgerknoten enthalten dieselbe Serverzuordnung für Aktivität 1 und alle möglichen Serverzuordnungen für Aktivität 2, usw. Die Blattknoten beschreiben dann eine vollständige Serverzuordnung für alle Aktivitäten $n \in N$. Der Test, ob ein Zweig weiter verfolgt werden muss, lautet wie folgt: Breche die Analyse eines Zweiges ab, wenn die Kosten einer Teillösung größer sind, als die der besten bisher gefundenen Gesamtlösung.

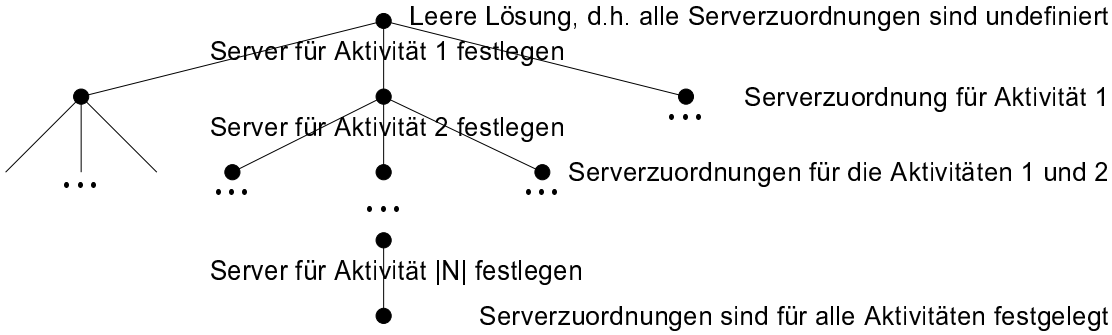


Abbildung 4.6 Berechnung der optimalen Serverzuordnungen dargestellt als Graphenproblem.

Der Einsatz des Backtracking-Verfahrens hat bei der gegebenen Problemstellung einen gravierenden Nachteil: Eine Teillösung verursacht i.d.R. geringere Kosten als eine Gesamtlösung, da nur die Ausführungs- und Migrationskosten eines Teils der Aktivitäten betrachtet werden. Deshalb müssen alle Zweige fast bis zum Ende durchlaufen werden, so dass annähernd so viele Kombinationen von

Serverzuordnungen analysiert werden müssen, wie bei Algorithmus 4.2. Der nachfolgend beschriebene Ansatz mindert dieses Problem etwas.

4.5.4.2 Branch-and-bound

Auch Branch-and-bound [BB93, LW66] ist ein Verfahren, um einen impliziten, gerichteten, azyklischen Graphen zu erkunden. Dabei wird intelligenter als beim Backtracking vorgegangen, indem versucht wird, auch die Kosten für den noch nicht feststehenden Teil der Lösung abzuschätzen.

Das Grundprinzip von Branch-and-bound ist dasselbe wie beim Backtracking. Allerdings wird für eine Teillösung i abgeschätzt, wie gut die Gesamtlösung bestenfalls noch werden kann ($w(i)$). Die möglichen Erweiterungen der Teillösung i werden in der Reihenfolge aufsteigender $w(i)$ durchlaufen, d.h. die vielversprechendste Erweiterung wird zuerst verfolgt. Kann eine Teillösung selbst im günstigsten Fall zu keiner besseren Gesamtlösung führen, als die beste schon gefundenen Gesamtlösung, so wird der entsprechende Zweig nicht weiter untersucht. Ein Beispiel, in dem Branch-and-bound eingesetzt werden kann, ist das Traveling-Salesman-Problem [BB93, Sch97a]. Da auch die (minimalen) Kosten des noch nicht feststehenden Teils der Lösung berücksichtigt werden, wird die Analyse „falscher“ Zweige früher abgebrochen, als beim Backtracking. Dadurch wird nur ein kleinerer Teil des Lösungsraums untersucht.

Um Branch-and-bound für das vorliegende Optimierungsproblem einsetzen zu können, muss abgeschätzt werden, wie hoch die Kosten einer Verteilung minimal sind, wobei die Serverzuordnungen nur teilweise bekannt sind. Für den Teil des Graphen, für den die Serverzuordnungen feststehen, ist die Berechnung der Kosten unproblematisch. Für Aktivitäten mit noch nicht feststehenden Serverzuordnungen ist es möglich, die minimalen Kosten für die Steuerung einzelner Aktivitäten zu berechnen (vgl. Phase 1 von Algorithmus 4.3). Die untere Schranke für die Kosten von Migrationen, in welche diese Aktivitäten involviert sind, beträgt 0, da stets der Fall eintreten kann, dass keine Migration stattfindet. Dadurch wird die Abschätzung relativ grob, so dass auch bei diesem Verfahren recht viele Knoten besucht werden müssen. I.Allg. entsteht ein exponentiell großer Aufwand, so dass Branch-and-bound nur in Ausnahmefällen eingesetzt werden kann. Insbesondere, da die in Abschnitt 4.5.2 und 4.5.3 vorgestellten Algorithmen für die praktisch relevanten Fälle sehr gute Lösungen ermitteln, macht es wenig Sinn, ein Branch-and-bound-Verfahren einzusetzen. Falls aber aus irgendwelchen Gründen die absolut optimalen Serverzuordnungen benötigt werden, so ist Branch-and-bound das am besten geeignete Verfahren, um diese zu ermitteln.

4.5.4.3 Hill-Climbing

Hill-Climbing ist ein Verfahren aus dem Bereich des maschinellen Lernens. Es verfolgt das Ziel, eine größere Anzahl von Variablen so zu konfigurieren, dass eine Zielfunktion minimiert wird. Es wird also die Lösung für ein Optimierungsproblem gesucht. Dabei wird aber im Gegensatz zu den vorherigen beiden Verfahren nicht immer das absolute Optimum ermittelt. Damit fällt dieses Verfahren in dieselbe Kategorie wie die Verteilungsalgorithmen aus Abschnitt 4.5.2 und 4.5.3.

Das klassische Hill-Climbing [Emd91] beginnt mit einer leeren Lösung. Mit einer Bewertungsfunktion werden alle elementaren Änderungen (z.B. Veränderungen einer einzelnen Variablen) analysiert. Dann wird diejenige Änderung, die mit der größten Verbesserung verbunden ist, durchgeführt. Ist keine Verbesserung mehr möglich, so stoppt der Algorithmus. Ein Problem dieses Verfahrens ist, dass

der Algorithmus auch in einem lokalen Optimum stoppen kann, das absolute Optimum also nicht gefunden wird.

Beim iterativen Hill-Climbing [Ack90, OvG89] wird mit einer zufälligen Belegung der Variablen begonnen. Dann wird wie beim Standardverfahren solange der „steilste Anstieg“ gewählt, bis ein lokales Optimum erreicht wird. Das gesamte Verfahren wird solange wiederholt, bis die vorgesehene Zeit abgelaufen ist. Durch zahlreiche Wiederholungen kann die Wahrscheinlichkeit reduziert werden, dass nur ein lokales Optimum gefunden wird. Dieses Verfahren hat wegen seiner Einfachheit eine sehr gute Performanz (siehe [Ack90]). Das iterative Hill-Climbing mit steilstem Anstieg kann noch modifiziert werden, indem nicht alle möglichen Veränderungen analysiert werden, sondern die erste gefundene Verbesserung umgesetzt wird („nächster Anstieg“). Dies reduziert die Anzahl der zu analysierenden Variablenbelegungen und damit den Zeitbedarf erheblich. Dies gilt insbesondere dann, wenn der Aufwand zur Berechnung der Bewertungsfunktion groß ist.

Beim stochastischen Hill-Climbing [Ack90, OvG89] sind auch Änderungen erlaubt, die mit einer Verschlechterungen der Zielfunktion einhergehen. Dadurch ist es möglich, aus lokalen Optima zu „entkommen“. Ob eine Veränderung akzeptiert wird, ist abhängig von der Größe der Verbesserung bzw. Verschlechterung und von einer (zu konfigurierenden) Konstanten T (Temperatur). Dabei ist die Wahrscheinlichkeit, dass große Verbesserungen (Verschlechterungen) akzeptiert werden, groß (klein). Die Temperatur[4] T gibt an, wie stark die Veränderungen gewichtet werden. Bei diesem Verfahren wird die jeweils beste bisher gefundene Lösung gespeichert und am Testende ausgegeben. Beim iterativen stochastische Hill-Climbing [OvG89] wird dieser Vorgang mit einer jeweils zufälligen Startbelegung der Variablen mehrfach durchgeführt und das Optimum aller Testläufe ermittelt.

Prinzipiell können mit allen Hill-Climbing-Varianten geeignete Serverzuordnungen für die Aktivitäten berechnet werden. Dabei ist aber von Nachteil, dass bei den Verfahren eine leere oder zufällige Startbelegung der Variablen (initiale Serverzuordnungen) verwendet werden muss. Es wäre wesentlich günstiger, wenn von den in Phase 1 von Algorithmus 4.3 berechneten optimalen Serverzuordnungen für einzelne Aktivitäten ausgegangen werden könnte. Da die Verfahren dies nicht zulassen, wird relativ „blind“ gesucht, was zu sehr großen Rechenzeiten führt. Da zudem der Suchraum sehr groß ist ($|N| > 100$ mit jeweils mehreren möglichen Serverzuordnungen, in [Ack90] wurde von 20 bis 30 binären Variablen ausgegangen) sind Hill-Climbing-Verfahren kaum einsetzbar.

Es ist aber durchaus sinnvoll, Hill-Climbing für die Phase 2 der Verteilungsalgorithmen einzusetzen, d.h. beim Zusammenfassen von Partitionen. In Algorithmus 4.3 wurde ein „nächster Anstieg“-Verfahren in Kombination mit einer Heuristik eingesetzt, die angibt, welche Schritte für das nächste Zusammenfassen in Frage kommen. Dadurch wird nicht „blind“ irgendeine gefundene Verbesserung gewählt, sondern es wird „vorsortiert“. Der Algorithmus 4.4 verwendet in seiner Phase 2 ein klassisches Hill-Climbing, indem immer diejenige Zusammenfassung ausgewählt wird, die zu der maximalen Verbesserung führt.

4.5.4.4 Simulated-Annealing

Simulated-Annealing ist eine Weiterentwicklung des Hill-Climbing. Es hat seine Ursprünge in der Physik [DS90, OvG89]: Bei hohen Temperaturen herrscht in Gasen viel Bewegung. Werden diese langsam abgekühlt, so erstarren sie zu einer (perfekten) Kristallstruktur. Diese entspricht einer optimalen Belegung der Variablen.

[4] Analogie zu Gasen: Bei hohen Temperaturen bewegen sich die Teilchen schnell. Bei niedrigen Temperaturen sind nur geringfügige Veränderungen möglich.

Beim Simulated-Annealing [Ack90, OvG89, Sch97a] wird ein Hill-Climbing-Verfahren eingesetzt. Dabei wird mit einer sehr hohen Temperatur begonnen. Deshalb ist am Anfang die Wahrscheinlichkeit sehr groß, dass eine Änderung (auch Verschlechterung) akzeptiert wird. Die Variablenbelegung springt stark. Im Laufe der Zeit wird die Temperatur abgesenkt. Dadurch werden die Schwankungen (insbesondere die Verschlechterungen) kleiner. Auf diese Weise nähert sich die Variablenbelegung einem Optimum. Die Gesamtlösung ergibt sich als die beste aller betrachteten Variablenbelegungen. Durch die starken Schwankungen am Anfang des Annealing-Prozesses, kann das Verfahren aus lokalen Optima entkommen. Deshalb ist es für zahlreiche Anwendungen geeignet und führt zu besseren Resultaten als Hill-Climbing [DS90]. Eine typische Anwendung für Simulated-Annealing ist die Planung des Verlaufs von Leiterbahnen in VLSI-Schaltkreisen.

Beim Simulated-Annealing ist es schwierig, den Verlauf der Temperatur T geeignet festzulegen (vgl. [Ack90]). Sinkt T zu langsam, so wird unnötig viel Rechenzeit verbraucht. Sinkt T zu schnell, so wird keine gute Lösung gefunden, weil eine noch ungünstige Belegung „eingefroren“ wird. Diese kann sich bei kleinen Werten für T kaum noch ändern. Dasselbe Problem ergibt sich für die Höhe des Startwerts von T: Ist er zu groß, so braucht das Abkühlen sehr lange. Ist er zu klein, so können lokale Minima nicht verlassen werden.

Ebenso wie Hill-Climbing kann Simulated-Annealing auf eine leere oder zufällig festgelegte Anfangsbelegung der Variablen (Serverzuordnungen) angewandt werden. Deshalb können mit diesem Verfahren geeignete Serverzuordnungen berechnet werden. Es macht dabei jedoch keinen Sinn von den in Phase 1 von Algorithmus 4.3 berechneten Serverzuordnungen auszugehen, da diese durch die großen Veränderungen am Beginn des Verfahren sehr schnell und beliebig verändert werden. Deshalb wird diese günstige Variablenbelegung nicht genutzt. Somit treten dieselben Nachteile wie beim Hill-Climbing auf, nämlich dass relativ „blind“ nach einer Lösung gesucht wird. Durch die Möglichkeit, lokale Optima zu verlassen, führt Simulated-Annealing aber i.d.R. zu besseren Ergebnissen als Hill-Climbing, falls der Verlauf der Temperatur T geeignet gewählt wird.

Die Anwendung von Simulated-Annealing auf das Zusammenfassen von Partitionen (Phase 2) ist problematisch. Vorraussetzung für die Verwendung von Simulated-Annealing ist generell, dass jeder Zustand von jedem anderen ausgehend erreichbar ist [OvG89]. Nur so können falsche Entscheidungen, die bei hohen Temperaturen häufig sind, ausgeglichen werden. Zusammenfassungen können aber durch weitere Zusammenfassungen nicht mehr rückgängig gemacht werden. Partitionen, die bei einer sehr hohen Temperatur unsinnigerweise zusammengefasst werden, bleiben also für den Rest des Annealing-Prozesses zusammen. Da es sehr viele Möglichkeiten zum unsinnigen Zusammenfassen gibt, wird dies bei hohen Temperaturen fast immer vorkommen, so dass die mehrfache Wiederholung des Annealing-Algorithmus auch keine Lösung darstellt. Deshalb ist eine Verwendung von Simulated-Annealing für das Zusammenfassen von Partitionen kaum sinnvoll.

4.6 Verteilung der Benutzer auf die Teilnetze

Ob für die Aktivitäten überhaupt günstige Serverzuordnungen existieren, hängt davon ab, wie die Benutzer auf die verschiedenen Teilnetze verteilt sind. Wie wir in Abschnitt 3.1.5 gesehen haben, ist es dabei vorteilhaft, wenn die Bearbeiter einer Aktivität eine möglichst hohe Lokalität zueinander aufweisen. Ist dies der Fall, so kann der WF-Server nahe bei allen diesen Bearbeitern gewählt werden. Sind andererseits die Bearbeiter einer Aktivität gleichmäßig über alle Teilnetze verteilt, so ist es gleichgültig, welcher WF-Server gewählt wird. Es gibt dann keinen besonders gut geeigneten WF-Server. Aus diesen Gründen sollten die Benutzer – sofern dies in der gegebenen Umgebung möglich

ist – den Teilnetzen so zugeordnet werden, dass die Bearbeiter einer Aktivität eine hohe Lokalität zueinander aufweisen. Eine solche Zuordnung zu finden ist nicht einfach, da es eine Vielzahl von Aktivitäten mit unterschiedlichen Bearbeiterzuordnungen und unterschiedlicher Relevanz gibt. Deshalb haben wir ein Verfahren entwickelt, das Mengen von Bearbeitern berechnet, die demselben Domain angehören sollten. Im Folgenden wird zuerst seine Anwendbarkeit diskutiert und dann wird es vorgestellt. Da bei dem Verfahren angenommen wird, dass sich stets alle Bearbeiter $u \in PotBearb_n$ für die Bearbeitung der Aktivität n qualifizieren, kann das Verfahren in der hier vorgestellten Form nur für unabhängige Bearbeiterzuordnungen verwendet werden.

4.6.1 Anwendbarkeit des Verfahrens

Damit überhaupt günstige Serverzuordnungen für die Aktivitäten festgelegt werden können, sollte die Verteilung der Benutzer auf die Domains optimiert werden. Dies ist aber leider nicht immer möglich. Häufig entstehen durch äußere Vorgaben Restriktionen bezüglich des einem Benutzer zugeordneten Teilnetzes. Deshalb wird nun diskutiert, wann das nachfolgend vorgestellte Verfahren einsetzbar ist.

Ein günstigster Fall ist, dass ein Computersystem komplett neu aufgebaut wird. Doch selbst dann entstehen durch die physische Anordnung der Benutzer (z.B. die Lage ihrer Büros) Restriktionen für das Kommunikationsnetzwerk. Es ist nicht möglich, einen Benutzer einem beliebigen Teilnetz zuzuordnen, da ein LAN nicht beliebig weit voneinander entfernte Benutzer verbinden kann (vgl. [Tan92, CDK95]). Man hat also nur in den äußerst seltenen Fällen, dass ein Gebäude neu bezogen wird oder ein Wechsel der Arbeitsplätze akzeptabel ist, wirklich alle Freiheitsgrade bei der Zuordnung der Benutzer zu den Teilnetzen. Ist dagegen die physische Anordnung der Benutzer vorgegeben, das Kommunikationsnetzwerk soll aber noch aufgebaut werden, so sind bestimmte Zuordnungen nicht möglich. Wie stark die Einschränkungen dann sind, hängt von der konkret verwendeten Netzwerktechnologie ab. Der Fall, bei dem am wenigsten Optimierungspotential existiert, ist, dass ein existierendes Kommunikationssystem weiterverwendet werden soll. Dann bestimmen die Möglichkeiten zur Änderung der Netzwerkkonfiguration die Freiheitsgrade bei der Optimierung.

Im Abschnitt 4.6.3 wird ein Algorithmus vorgestellt, der einen Vorschlag für eine geeignete Zuordnung der Benutzer des WfMS zu den Domains ermittelt. Dieser kann übernommen werden, falls ausreichend Freiheitsgrade vorhanden sind. Andernfalls kann er als Anregung für Optimierungen dienen. Um diese zu realisieren, können z.B. einzelne Benutzer einem anderen Teilnetz (Domain) zugeordnet werden oder es kann ein zusätzliches Teilnetz verwendet werden. In Fällen, in denen überhaupt keine Freiheitsgrade bestehen, d.h., wenn der WF-Administrator keinen Einfluss auf die Netzwerkplanung hat, kann das vorgestellte Verfahren nicht eingesetzt werden. Dem WfMS wird dann nur die aktuelle Zuordnung der Benutzer zu den Teilnetzen vorgegeben, da diese für das Kostenmodell benötigt wird. Doch auch ohne zusätzliche Optimierung ist eine solche Zuordnung häufig schon recht günstig, da Benutzer mit ähnlichen Aufgaben normalerweise derselben OE angehören und deshalb eine hohe Lokalität zueinander aufweisen.

4.6.2 Partitionierung in verteilten Datenbanksystemen

Die Verteilung der Daten eines verteilten DB-Systems auf die einzelnen Teilsysteme wird an dieser Stelle betrachtet, weil aus diesem Forschungsgebiet stammende Algorithmen auf unser Problem übertragbar sind. Deshalb werden die entsprechenden Verfahren im Folgenden kurz vorgestellt. Im

nächsten Abschnitt wird dann gezeigt, wie sie verwendet werden können, um die optimalen Teilnetze für die Benutzer zu berechnen.

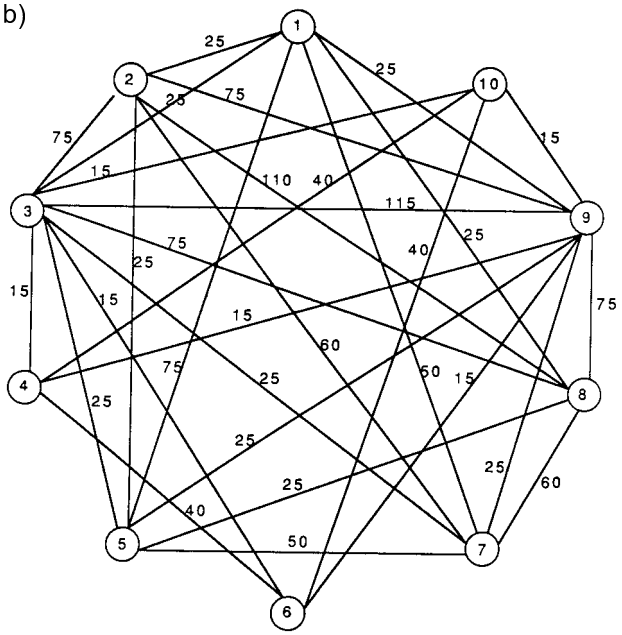
In verteilten DB-Systemen muss eine geeignete Verteilung der Daten auf die Teilsysteme gefunden werden [Dad96, ML77, ÖV91], um effizient arbeiten zu können. Das dafür übliche Verfahren ist die horizontale Partitionierung, bei dem die Tupel einer Relation mittels Prädikaten unterschiedlichen DB-Servern zugeordnet werden. Dieser Ansatz ist zur Lösung unserer Fragestellung nicht geeignet. Die vertikale Partitionierung wird weniger häufig verwendet, ist aber algorithmisch interessanter. Hierbei werden die verschiedenen Attribute einer Relation in unterschiedlichen Teilsystemen gespeichert. Welches Teilsystem für ein Attribut gewählt wird, ist abhängig davon, wo das Attribut am häufigsten verwendet wird. Verfahren, mit denen eine solche Verteilung berechnet werden kann, wollen wir nun näher betrachten.

In [NCWD84, NR89] werden Verfahren zur vertikalen Partitionierung von Relationen vorgestellt. Dabei wird von einer Attributs-Affinitäts-Matrix ausgegangen, die angibt, wie groß die Affinität zwischen Paaren von Attributen ist. Um die Attributs-Affinitäts-Matrix zu ermitteln, wird analysiert, welche Attribute in welchen Transaktionen verwendet werden und wie häufig diese Transaktionen sind. Werden zwei Attribute häufig in derselben Transaktion verwendet, so besitzen sie eine große Affinität, d.h. einen hohen Wert in der Attributs-Affinitäts-Matrix. In Abb. 4.7a ist ein Beispiel für eine Attributs-Affinitäts-Matrix dargestellt.

a)

	1	2	3	4	5	6	7	8	9	10
1	75	25	25	0	75	0	50	25	25	0
2	25	110	75	0	25	0	60	110	75	0
3	25	75	115	15	25	15	25	75	115	15
4	0	0	15	40	0	40	0	0	15	40
5	75	25	25	0	75	0	50	25	25	0
6	0	0	15	40	0	40	0	0	15	40
7	50	60	25	0	50	0	85	60	25	0
8	25	110	75	0	25	0	60	110	75	0
9	25	75	115	15	25	15	25	75	115	15
10	0	0	15	40	0	40	0	0	15	40

b)



c) Cluster 1: 1, 5, 7
Cluster 2: 2, 3, 8, 9
Cluster 3: 4, 6, 10

Abbildung 4.7 Beispiel aus [NR89] für a) eine Attributs-Affinitäts-Matrix, b) ihre Darstellung als Graph (ohne Kanten mit dem Wert 0) und c) die resultierenden Cluster.

Im Folgenden werden zwei verschiedene Verfahren vorgestellt und verglichen, mit denen ermittelt werden kann, welche Attribute demselben Teilsystem zugeordnet werden sollen. Dabei wird auch geprüft, ob sie zur Berechnung einer geeigneten die Verteilung der Benutzer eines WfMS verwendet werden können. In [NCWD84] wird ein Verfahren vorgestellt, das eine Abschätzung der durch eine gewählte Partitionierung entstehenden Kosten und die Festlegung einer Zielfunktion erfordert. Beides ist äußerst kritisch, da die Qualität des ermittelten Ergebnisses davon abhängt. Dadurch ist es schwie-

rig, den Ansatz auf unser Problem zu übertragen. Der in [NR89] vorgestellte Ansatz ist dafür besser geeignet. Dieser sucht in einem Graphen nach Mengen von Knoten (Attributen) mit einer starken wechselseitigen Verbindung. Für diesen Ansatz wird keine Zielfunktion und keine Abschätzung der entstehenden Kosten benötigt. Außerdem hat er mit $O(\#Knoten^2)$ die bessere Komplexität. Deshalb ist es empfehlenswert, den Algorithmus aus [NR89] zu verwenden. Der aus der Attributs-Affinitäts-Matrix aus Abb. 4.7 erzeugte Graph und die vom Algorithmus ermittelten Mengen (Cluster) sind in Abb. 4.7b und c dargestellt.

Auf eine detaillierte Beschreibung des Algorithmus wird hier verzichtet. Eine solche findet sich außer in [NR89] auch in [Boc96, End98]. Das Verfahren soll aber kurz skizziert werden. Es beginnt mit einem beliebigen („aktuellen“) Knoten des Graphen. Vereinfacht gesprochen wählt der Algorithmus dann denjenigen Knoten aus, der die größte Affinität zum aktuellen Knoten hat. Dann testet er, ob dieser Knoten in den Cluster aufgenommen werden darf. Dies ist erlaubt, falls dadurch das Affinitätsniveau des Clusters nicht gesenkt wird. Das Affinitätsniveau eines Clusters ist als die niedrigste Affinität definiert, die zwischen zwei Elementen des Clusters besteht. Dieser Vorgang wird solange wiederholt, bis keine weiteren Knoten mehr in den Cluster aufgenommen werden können. Tritt dieser Fall ein, so setzt der Algorithmus seine Arbeit mit einem anderen Knoten fort, wodurch ein weiterer Cluster gebildet wird. Das Verfahren berechnet also Cluster von Knoten, die zueinander eine hohe Affinität haben. In [NR89] wird bewiesen, dass es keine Rolle spielt, mit welchem Knoten das Verfahren beginnt. Das erzielte Ergebnis hat stets die gleiche Qualität, was zeigt, dass das Verfahren „sinnvolle Cluster“ ermittelt.

4.6.3 Berechnung der optimalen Teilnetze für die Benutzer

Die im vorherigen Abschnitt vorgestellten Verfahren können zum Ermitteln von Mengen solcher Benutzer verwendet werden, welche dieselben Aktivitäten bearbeiten und deshalb demselben Domain angehören sollten. Allerdings sind hierfür einige zusätzliche Schritte notwendig, die im Folgenden vorgestellt werden (siehe auch [BD97]).

Der Algorithmus 4.6 erstellt zuerst eine Matrix $user\_activity$, in der für jede Aktivität n, die der Benutzer u ausführen darf, sein Gewicht $G_n(u)$ übernommen wird. Ein Beispiel für eine solche Matrix findet sich in Abb. 4.8b. Diese wird so zu $user\_activity'$ normiert, dass die Summe jeder Zeile 1 beträgt. Der Teiler ist bei dieser Berechnung stets ungleich 0, da $\forall u \in U : user\_activity_{n,u} = 0$ bedeuten würde, dass kein Benutzer die Aktivität n bearbeiten darf. Falls es solche Aktivitäten n geben sollte, so können sie bei dieser Berechnung ignoriert werden. Damit gibt der Eintrag $user\_activity'_{n,u}$ die Wahrscheinlichkeit an, dass die Aktivität n vom Benutzer u bearbeitet wird[5] (vgl. Abb. 4.8c). Mit dieser Information kann jetzt die Affinitätsmatrix berechnet werden. Ein Benutzerpaar hat in dieser Matrix einen hohen Wert, wenn die Wahrscheinlichkeit groß ist, dass sich beide Benutzer für die Ausführung derselben Aktivität qualifizieren. Für die Benutzer u_1 und u_2 und die Aktivität n beträgt diese Wahrscheinlichkeit $user\_activity'_{n,u_1} \cdot user\_activity'_{n,u_2}$. Um die Relevanz der verschiedenen Aktivitäten zu berücksichtigen, wird diese Wahrscheinlichkeit noch mit der Ausführungshäufigkeit der Aktivität n (siehe Abb. 4.8a) gewichtet, so dass sich die in Abb. 4.8d dargestellte Affinitätsmatrix ergibt. Diese Matrix dient nun als Eingabe für die in Abschnitt 4.6.2 vorgestellten Algorithmen. Im Beispiel aus Abb. 4.8 ergibt sich dadurch das Ergebnis $Cluster = \{\{1, 4\}, \{2, 3\}\}$, d.h. die Benutzer 1 und 4 sollten demselben Teilnetz zugeordnet werden, und ebenso die Benutzer 2 und 3.

[5] Dabei wird angenommen, dass die „Auswahl“ des Bearbeiters einer Aktivität unabhängig erfolgt. Dies ist gegeben, da eventuelle Ungleichverteilungen schon durch die Gewichte der Benutzer berücksichtigt sind (siehe Abschnitt 4.2.2.2).

a)

n	1	2	3
NodeFreq(n)	100	60	10

b)

	user 1	2	3	4
step 1	1	0	0	0,5
step 2	0	1	1	0
step 3	0	1	1	0,5

c)

	user 1	2	3	4
step 1	2/3	0	0	1/3
step 2	0	1/2	1/2	0
step 3	0	2/5	2/5	1/5

d)

	1	2	3	4
1	44,4	0	0	22,2
2	0	16,6	16,6	0,8
3	0	16,6	16,6	0,8
4	22,2	0,8	0,8	11,5

Abbildung 4.8 Beispiel für a) Ausführungshäufigkeiten von Aktivitäten *NodeFreq*, b) eine Matrix $user\_activity$, c) die zugehörige normierte Darstellung $user\_activity'$ und d) die daraus resultierende Affinitätsmatrix *affinity*.

Algorithmus 4.6 (Berechnung von Clustern von Benutzern)
input
U: die Menge der Benutzer des WfMS
N: die Menge aller Aktivitäten aller WF-Vorlagen
$PotBearb_n$: $\forall n \in N$: die potentiellen Bearbeiter der Aktivität n
$G_n(u)$: $\forall u \in U, \forall n \in N$: Gewicht des Benutzers u bei der Ausführung von Aktivität n
$NodeFreq(n)$: $\forall n \in N$: Ausführungshäufigkeit von Aktivitäteninstanzen von Typ n
output
$Cluster$: Menge von Clustern aus Benutzern, die demselben Domain angehören sollen
begin
// erzeuge die Matrix $user\_step$
for each $n \in N$ **do**
for each $u \in U$ **do**
if $u \in PotBearb_n$ **then**
$user\_step_{n,u} = G_n(u)$;
else
$user\_step_{n,u} = 0$;
// normalisiere die Matrix $user\_step$ zu $user\_step'$
for each $n \in N$ **do**
for each $u \in U$ **do**
$user\_step'_{n,u} = user\_step_{n,u} \;/\; \sum_{u' \in U} user\_step_{n,u'}$;
// berechne die Benutzer-Affinitätsmatrix *affinity*
for each $u_1 \in U$ **do**
for each $u_2 \in U$ **do**
$affinity_{u_1,u_2} = \sum_n user\_step'_{n,u_1} \cdot user\_step'_{n,u_2} \cdot NodeFreq(n)$;
// berechne Cluster der Benutzer $Cluster$
Algorithmen aus Abschnitt 4.6.2 auf *affinity* anwenden,
Ergebnis: $Cluster = \{cluster_1, cluster_2, \ldots\}$;
end.

Der Algorithmus 4.6 berechnet Mengen von Benutzern, die ähnliche Aufgaben erledigen, ohne die „Qualität“ der ermittelten Cluster zu betrachten. Das heißt, es kann vorkommen, dass ein Cluster zu groß ist. Dies würde zu einer Überlastung des entsprechenden Teilnetzes führen. In diesem Fall muss der Cluster vom WF-Modellierer aufgespalten werden. Falls die Anzahl der ermittelten Cluster größer ist, als die Anzahl der zur Verfügung stehenden Teilnetze, so müssen die Benutzer mehrerer (kleiner) Cluster demselben Teilnetz zugeordnet werden. Außerdem kann es vorkommen, dass die physische Anordnung der Benutzer verhindert, dass bestimmte Benutzer demselben Teilnetz zugeordnet werden (vgl. Abschnitt 4.6.1). In diesen Fällen müssen einige dieser Benutzer anderen Clustern zugeordnet werden.

4.7 Analyse der Effizienz des Verfahrens

In diesem Abschnitt wird untersucht, in wie weit durch die verteilte WF-Ausführung in ADEPT eine Verbesserung der Leistungsfähigkeit des WfMS erreicht wird. Dazu werden aus dem Bereich der Parallelrechner bekannte Beschränkungen untersucht und mit dem bei unserem Verteilungsmodell vorliegenden Szenario verglichen. Anschließend wird analysiert, welche Auswirkungen Migrationen auf die durch die verteilte WF-Steuerung erreichbare Leistungssteigerung haben. Außerdem wird gezeigt, dass Migrationen sogar dazu beitragen können, die insgesamt anfallenden Kommunikationskosten zu reduzieren.

4.7.1 Grenzen der Parallelverarbeitung

Aus dem Bereich massiv paralleler Rechner sind verschiedene Grenzen für die durch Parallelisierung erreichbare Leistungssteigerung bekannt. Wir wollen diese im Folgenden vorstellen. Anschließend wird untersucht, ob dieselben Beschränkungen für das Verteilungsmodell von ADEPT gelten. Zu diesem Zweck definieren wir zuerst den Begriff des Speedup S_n. Dies ist der Faktor, um den ein Problem schneller gelöst wird, wenn n Prozessoren verwendet werden.

Definition 4.1 (Speedup nach [Bem92])
Sei T_1 bzw. T_n die Berechnungszeit eines gegebenen Problems auf 1 bzw. n Prozessoren. Dann ist Speedup S_n definiert als $S_n = T_1/T_n$

4.7.1.1 Groschs Gesetz

Groschs Gesetz [Qui88, Gro53, Gro75] besagt, dass die Geschwindigkeit von Computern quadratisch mit den dafür entstehenden Kosten wächst. Deshalb macht es keinen Sinn, mehrere kleine Rechner zu vernetzen, um diese eine Aufgabe gemeinsam lösen zu lassen. Es ist billiger, einen großen Rechner zu kaufen, der so viel Rechenleistung besitzt, wie die kleinen Rechner zusammen.

In [Qui88] wird darauf hingewiesen, dass diese Regel nur innerhalb einer Rechnerklasse (Mainframes, Minirechner, Mikrorechner) gilt. Es ist durchaus möglich, dass mehrere Minirechner, die zusammen dieselbe Rechenleistung haben wie ein Mainframe, trotzdem billiger sind als dieser.

4.7.1.2 Minskys Hypothese

Minskys Hypothese [Qui88, Bem92, MP71] vermutet, dass durch die Verwendung von Parallelrechnern maximal eine Beschleunigung erreicht werden kann, die logarithmisch zur Anzahl der verwendeten Prozessoren ist. Das heißt, für den Speedup gilt: $S_n \leq \log n$. Deshalb wird Parallelverarbeitung ab einer gewissen Grenze ineffizient.

Diese Annahme stellt eine sehr starke Beschränkung dar. Es hängt aber vom konkreten Algorithmus ab, ob die Hypothese tatsächlich zutrifft und wie stark die Beschränkung ist. Bei Parallelrechnern ist der erreichbare Speedup außerdem von der Rechnerarchitektur abhängig. Es gibt Algorithmen, bei denen bei über 100 Prozessoren eine lineare Beschleunigung erreicht werden kann [Qui88].

4.7.1.3 Amdahls Gesetz

Amdahls Gesetz [Qui88, Bem92, Amd67] macht Aussagen über den maximal möglichen Speedup in Abhängigkeit des nicht parallelisierbaren Anteils eines Problems. Angenommen wird, dass jeder Algorithmus einen Anteil von f ($0 \leq f \leq 1$) nicht parallelisierbarer Operationen beinhaltet, d.h. Operationen, die von einem Prozessor ausgeführt werden müssen, während die anderen Prozessoren auf dessen Ergebnis warten. Dann lässt sich der bei n Prozessoren maximal erreichbare Speedup wie folgt berechnen:

$$S_n \leq \frac{1}{f + (1-f)/n}$$

Damit ist S_n durch $1/f$ beschränkt, selbst wenn beliebig viele Prozessoren verwendet werden. Enthält ein Programm 10% nicht parallelisierbare Operationen, so beträgt der maximal durch Parallelität erreichbare Speedup 10. Aus Amdahls Gesetz kann nun geschlossen werden, dass es keinen Sinn macht, sehr viele Prozessoren zu verwenden, da ab einer gewissen Grenze keine weitere Verbesserung mehr erreicht werden kann.

4.7.1.4 Lees Schranke

In Lees Schranke [Bem92, Lee77] wird zusätzlich berücksichtigt, dass es selbst im parallelen Anteil der Algorithmen i.d.R. nicht möglich ist, alle Prozessoren auszulasten. Damit wird bei n Prozessoren der folgende maximale Speedup errechnet:

$$S_n \leq \frac{n}{\log n}$$

Dies stellt eine weniger starke Beschränkung dar, als bei Minskys Hypothese mit $\log n$ vermutet.

4.7.1.5 Diskussion

Wir wollen nun untersuchen, in wie weit die vorgestellten Schranken aus der Welt der Parallelrechner für die Effizienz des Verteilungsmodells von ADEPT von Bedeutung sind. Groschs Gesetz sagt aus, dass es billiger ist, einen leistungsstarken Rechner zu verwenden, als mehrere kleinere Rechner. Folgt man dieser Regel, so darf kein verteiltes WfMS realisiert werden. Stattdessen muss ein einziger leistungsstarker zentraler WF-Server verwendet werden. Allerdings gilt Groschs Gesetz nicht bei beliebiger Leistung: Es gibt für jede Rechnerklasse einen Rechner mit der zur Zeit maximal verfügbaren Leistung und dieser hat einen bestimmten Preis. Es ist nicht möglich, einen Rechner mit noch höherer Leistung zu kaufen. In diesem Fall kann man nur mehr Leistung für die WF-Server erhalten, indem mehrere Rechner verwendet werden, oder indem ein Rechner einer anderen Rechnerklasse ausgewählt wird, was wiederum höhere Kosten verursacht. Außerdem sind bei unternehmensweiten WfMS die Benutzer (weiträumig) in einem Kommunikationssystem verteilt. In einer solchen Umgebung werden schon deshalb mehrere WF-Server benötigt, damit diese nahe bei den Benutzern angesiedelt sein können. Ansonsten würde man schlechte Antwortzeiten, hohe Kommunikationskosten und einen Engpass im Kommunikationssystem erhalten (vgl. Abschnitt 3.1).

Es bleibt noch zu untersuchen, ob die in Abschnitt 4.7.1.2 - 4.7.1.4 erwähnten Grenzen für den Speedup eine Restriktion für eine verteilte WF-Steuerung darstellen. Die dadurch vorgegebenen Schranken wären allerdings nicht besonders gravierend, da die Anzahl der WF-Server (maximal einige Dutzend) im Vergleich zu der Anzahl der Prozessoren in massiv parallelen Systemen (tausende

oder noch mehr) verhältnismäßig klein ist. In [Qui88] wird ausgesagt, dass es vom konkreten Algorithmus abhängt, ob diese Schranken wirklich zutreffen. Im nächsten Abschnitt wird gezeigt, dass dies in ADEPT nicht der Fall ist.

4.7.2 Parallelverarbeitung in ADEPT

Das Verteilungsmodell von ADEPT wurde so entworfen, dass keine zentralen Komponenten benötigt werden. Weil die Algorithmen deshalb keinen nicht parallelisierbaren Anteil beinhalten, treten die im vorherigen Abschnitt für Parallelrechner beschriebenen Effekte nicht auf. Lediglich bei Migrationen werden zwei WF-Server synchronisiert, wohingegen die anderen weiterarbeiten können. In einem verteilten WfMS wird (im Gegensatz zu Parallelrechnersystemen) nicht nur eine einzelne Aufgabe bearbeitet, sondern viele Einzelaufgaben. Die Anzahl der bearbeiteten WF-Instanzen (und erst recht der Aktivitäten) ist viel größer als die Anzahl der WF-Server. Da deshalb immer ausreichend viele Aufgaben zur Bearbeitung anstehen, können stets alle WF-Server gleichzeitig arbeiten, so dass durch die Verteilung ein höherer Systemdurchsatz erreicht wird. Weil dadurch mehr Benutzer bedient werden können, wurde eines der mit der verteilten WF-Steuerung verfolgten wesentlichen Ziele erreicht.

Das andere wichtige Ziel dieser Arbeit ist die Reduzierung der Kommunikationslast. Inwieweit dies durch die verteilte WF-Steuerung erreicht werden kann, hängt vom konkreten WF ab. Insbesondere die Anzahl der Migrationen ist dafür ausschlaggebend, da diese zusätzliche Kommunikationen verursachen. Dahingegen werden die für die Steuerung einzelner Aktivitäten entstehenden Kommunikationskosten durch die verteilte WF-Ausführung reduziert, weil die Lokation der WF-Server optimiert wird. Deshalb betrachten wir im Folgenden ein Beispiel ohne und eines mit Migrationen.

4.7.2.1 Workflow-Ausführung ohne Migrationen

Der günstigste Fall, der bei einem verteilten WfMS eintreten kann, ist, dass jedem WF nur Bearbeiter aus einem einzigen Domain zugeordnet sind. Wird ein WF dem Server dieses Domains zugeordnet, so laufen WF-Instanzen unterschiedlicher Typen völlig unabhängig voneinander ab. Das heißt, in jeden WF ist nur ein WF-Server involviert und es gibt keine domainübergreifende Kommunikation. In einem solchen Szenario ist es offensichtlich, dass die in Abschnitt 4.7.1 vorgestellten Beschränkungen nicht gelten, und dass für die WF-Server durch die „verteilte WF-Ausführung“ kein Zusatzaufwand entsteht. Da keine Migrationen stattfinden, entsteht auch kein zusätzlicher Kommunikationsaufwand.

Gehören die Bearbeiter eines WF-Typs nur größtenteils demselben Domain an, so kann wiederum auf Migrationen verzichtet werden. Die WF-Server arbeiten unabhängig voneinander und es findet nur wenig teilnetzübergreifende Kommunikation statt. Auch in diesem Fall entsteht kein Zusatzaufwand für die WF-Server.

4.7.2.2 Workflow-Ausführung mit Migrationen

Betrachten wir nun den wesentlich interessanteren Fall, dass Migrationen stattfinden. Durch diese Migrationen wird Zusatzaufwand erzeugt, welcher jedoch nur die WF-Server und Teilnetze der beiden betroffenen Domains belastet. Wird die Anzahl der Domains erhöht, so werden weiterhin nur zwei WF-Server damit belastet, weshalb der durchschnittliche Zusatzaufwand pro WF-Server sinkt. Dies zeigt, dass die in Abschnitt 4.7.1 vorgestellten Beschränkungen auch dann nicht gelten, wenn Migrationen notwendig sind. Im Folgenden wird noch gezeigt (siehe auch [BD97]), dass es möglich ist,

durch Migrationen die Belastung des Kommunikationssystems zu reduzieren. Die Belastung der WF-Server steigt natürlich durch den für die Migrationen zusätzlich nötigen Aufwand. Wie groß dieser Zusatzaufwand für die WF-Server tatsächlich ist, wird in Kapitel 9 analysiert.

Gegeben sei das folgende Szenario: Ein WF beginnt mit den Aktivitäten N_1, die durch den WF-Server 1 kontrolliert werden. Auf diese folgen die Aktivitäten N_2, deren Bearbeiter überwiegend dem Domain 2 angehören. Die Frage ist nun, ob sich eine Migration zum WF-Server 2 lohnt, oder ob die Teilnetze weniger belastet werden, wenn diese Aktivitäten ebenfalls vom WF-Server 1 kontrolliert werden. Zur Berechnung der entstehenden Kommunikationskosten verwenden wir das Kostenmodell aus Abschnitt 4.3. Mit diesem werden die Kosten verglichen, die ohne bzw. mit Migration entstehen. Da die Kosten, die bei der Ausführung der Aktivitäten N_1 entstehen, in beiden Fallen identisch sind, werden sie hier nicht betrachtet. Es werden also nur die Kosten für die Steuerung der Aktivitäten N_2 und der evtl. notwendigen Migration berechnet.

Bei der Berechnung wird angenommen, dass sich $\#User_n(1)$ Bearbeiter aus Domain 1 für die Ausführung der Aktivität n qualifizieren und $\#User_n(2)$ Bearbeiter aus dem Domain 2. In den anderen Domains soll es keine geeigneten Bearbeiter geben. Sie würden das Ergebnis der Berechnung auch nicht beeinflussen, da durch diese in beiden Fällen gleich hohe Kosten entstehen. Die Gesamtzahl der Bearbeiter der Aktivität n ergibt sich als $\#User_n = \#User_n(1) + \#User_n(2)$. Da bei Verwendung von statischen Serverzuordnungen die Anzahl der potentiellen Bearbeiter einer Aktivität unabhängig vom gewählten Server ist, gilt für alle Server j: $\#User_n(1|j) = \#User_n(1)$ und $\#User_n(2|j) = \#User_n(2)$.

Analysieren wir zuerst den Fall ohne Migration, d.h. die Aktivitäten N_2 werden vom WF-Server 1 kontrolliert. Dann entstehen im Teilnetz 1 (dem des WF-Servers 1) für die Ausführung einer Aktivität $n \in N_2$ die folgenden Kosten:

$$
\begin{aligned}
&Vol^{Akt}_{TN,1}(n) \\
&= \Big(\underbrace{\sum_j ExProb_n(1,j)}_{\substack{\text{= 1, da Server} \\ \text{im Teilnetz 1}}} + \underbrace{\sum_{j\neq 1} ExProb_n(j,1)}_{\substack{\text{= 0, da Server nie in} \\ \text{einem Teilnetz } j\neq 1}}\Big) \cdot (In\_parameter\_size_n + Out\_parameter\_size_n) \\
&= In\_parameter\_size_n + Out\_parameter\_size_n
\end{aligned}
$$

Im Teilnetz 2 entstehen die Kosten:

$$
\begin{aligned}
&Vol^{Akt}_{TN,2}(n) \\
&= \Big(\underbrace{\sum_j ExProb_n(2,j)}_{\substack{\text{= 0, da Server} \\ \text{nicht im Teilnetz 2}}} + \underbrace{\sum_{j\neq 2} ExProb_n(j,2)}_{\substack{\neq 0, \text{ falls } j=1, \text{ entspricht} \\ \text{Anteil Bearb. in Teilnetz 2}}}\Big) \cdot (In\_parameter\_size_n + Out\_parameter\_size_n) \\
&= (\#User_n(2|1) \;/\; \sum_l \#User_n(l|1)) \cdot (In\_parameter\_size_n + Out\_parameter\_size_n) \\
&= (\#User_n(2)/\#User_n) \cdot (In\_parameter\_size_n + Out\_parameter\_size_n)
\end{aligned}
$$

Die Kosten für das Aktualisieren der Arbeitslisten berechnen sich wie folgt:

$$Vol^{AL}_{TN,1}(n)$$

$$= \Big(\underbrace{\sum_j ExProb_n(1,j)}_{\text{= 1, da Server im Teilnetz 1}} \cdot \sum_k \#User_n(k|1) + \sum_{l \neq 1} \cdot \sum_j \underbrace{ExProb_n(l,j)}_{\text{= 0, da Server nie in einem Teilnetz } l \neq 1} \cdot \#User_n(1|l)\Big)$$
$$\cdot (WL\_insert\_size_n + WL\_delete\_size_n)$$
$$= \sum_k \#User_n(k|1) \cdot (WL\_insert\_size_n + WL\_delete\_size_n)$$
$$= \#User_n \cdot (WL\_insert\_size_n + WL\_delete\_size_n)$$

$$Vol^{AL}_{TN,2}(n)$$
$$= \Big(\underbrace{\sum_j ExProb_n(2,j)}_{\text{= 0, da Server nicht im Teilnetz 2}} \cdot \sum_k \#User_n(k|1) + \sum_{l \neq 2} \cdot \sum_j \underbrace{ExProb_n(l,j)}_{\neq 0, \text{ falls } l=1} \cdot \#User_n(2|l)\Big)$$
$$\cdot (WL\_insert\_size_n + WL\_delete\_size_n)$$
$$= \underbrace{\sum_j ExProb_n(1,j)}_{\text{= 1, da Server im Teilnetz 1}} \cdot \#User_n(2|1) \cdot (WL\_insert\_size_n + WL\_delete\_size_n)$$
$$= \#User_n(2) \cdot (WL\_insert\_size_n + WL\_delete\_size_n)$$

Da die Aktivitäten N_1 und N_2 vom WF-Server 1 kontrolliert werden, treten keine Migrationskosten auf. Die Kommunikation der Aktivitätenprogramme mit externen Datenquellen ist unabhängig von der Lage des WF-Servers, weshalb sie hier nicht betrachtet wird. Damit ergibt sich in den Teilnetzen insgesamt die folgende Last für die Ausführung der Aktivitäten N_2:

$$L_1 = \sum_{n \in N_2} Vol^{Akt}_{TN,1}(n) + Vol^{Akt}_{TN,2}(n) + Vol^{AL}_{TN,1}(n) + Vol^{AL}_{TN,2}(n)$$
$$= \sum_{n \in N_2} In\_parameter\_size_n + Out\_parameter\_size_n$$
$$+ (\#User_n(2)/\#User_n) \cdot (In\_parameter\_size_n + Out\_parameter\_size_n)$$
$$+ \#User_n \cdot (WL\_insert\_size_n + WL\_delete\_size_n)$$
$$+ \#User_n(2) \cdot (WL\_insert\_size_n + WL\_delete\_size_n)$$

Falls die Aktivitäten N_2 von WF-Server 2 kontrolliert werden, ergibt sich analog (durch Vertauschen der Zahlen 1 und 2):

$$Vol^{Akt}_{TN,1}(n) = (\#User_n(1)/\#User_n) \cdot (In\_parameter\_size_n + Out\_parameter\_size_n)$$

$$Vol^{Akt}_{TN,2}(n) = In\_parameter\_size_n + Out\_parameter\_size_n$$

$$Vol^{AL}_{TN,1}(n) = \#User_n(1) \cdot (WL\_insert\_size_n + WL\_delete\_size_n)$$

$$Vol^{AL}_{TN,2}(n) = \#User_n \cdot (WL\_insert\_size_n + WL\_delete\_size_n)$$

Da die Aktivitäten N_1 vom WF-Server 1 kontrolliert werden und die Aktivitäten vom Server 2, findet von der letzten Aktivität n_1 von N_1 zur ersten Aktivität n_2 von N_2 eine Migration statt. Diese erzeugt im Teilnetz 1 die folgenden Kosten:

$$Vol^{Migr}_{TN,1}(n_1,n_2) = \sum_{j\neq 1}\Big(\underbrace{MigrProb_{n_1,n_2}(1,j)}_{\substack{=1,\text{ falls } j=2\\ \text{sonst } =0}} + \underbrace{MigrProb_{n_1,n_2}(j,1)}_{\substack{=0,\text{ da nicht zum Server 1}\\ \text{migriert wird}}}\Big)\cdot Migration\_size_{n_1,n_2}$$
$$= Migration\_size_{m,n}$$

Im Teilnetz 2 entstehen dieselben Kosten durch die Migration:

$$Vol^{Migr}_{TN,2}(n_1,n_2) = Vol^{Migr}_{TN,1}(n_1,n_2) = Migration\_size_{n_1,n_2}$$

Damit ergeben sich, wenn die Aktivitäten N_2 von Server 2 kontrolliert werden, die folgenden Kosten für die Teilnetze:

$$\begin{aligned} L_2 = {} & 2\cdot Migration\_size_{n_1,n_2} \\ & + \sum_{n\in N_2} (\#User_n(1)/\#User_n)\cdot(In\_parameter\_size_n + Out\_parameter\_size_n) \\ & \quad + In\_parameter\_size_n + Out\_parameter\_size_n \\ & \quad + \#User_n(1)\cdot(WL\_insert\_size_n + WL\_delete\_size_n) \\ & \quad + \#User_n\cdot(WL\_insert\_size_n + WL\_delete\_size_n) \end{aligned}$$

Die Migration vom Server 1 zum Server 2 lohnt sich bzgl. der entstehenden Kommunikationskosten, wenn $L_2 < L_1$, d.h. $L_1 - L_2 > 0$. Nachdem identische Terme entfernt wurden, ergibt sich:

$$\begin{aligned} & L_1 - L_2 = \\ & \sum_{n\in N_2}\Big((\#User_n(2)/\#User_n - \#User_n(1)/\#User_n)\cdot(In\_parameter\_size_n + Out\_parameter\_size_n) \\ & \quad + (\#User_n(2) - \#User_n(1))\cdot(WL\_insert\_size_n + WL\_delete\_size_n)\Big) \\ & - 2\cdot Migration\_size_{n_1,n_2} \end{aligned}$$

$L_1 - L_2$ ist damit genau dann > 0, wenn:

$$\begin{aligned} & 2\cdot Migration\_size_{n_1,n_2} \quad < \quad (\#User_n(2) - \#User_n(1)) \\ & \cdot \sum_{n\in N_2}\Big(\frac{In\_parameter\_size_n + Out\_parameter\_size_n}{\#User_n} + WL\_insert\_size_n + WL\_delete\_size_n\Big) \end{aligned}$$

Im Folgenden wollen wir annehmen, dass N_2 lauter gleichartige Aktivitäten enthält. Unter dieser Annahme wird untersucht, ab welcher Anzahl von Aktivitäten sich eine Migration lohnt. Falls $\#User_n(2) < \#User_n(1)$ gilt, ist die rechte Seite dieser Ungleichung negativ, so dass sie nie erfüllt werden kann. Das bedeutet, dass es sich in diesem Fall niemals lohnt, zu migrieren. Dies ist auch klar, da von einem gut geeigneten Domain in einen ungünstigen migriert werden würde. Gilt $\#User_n(2) > \#User_n(1)$, so ist die Ungleichung erfüllt, wenn die Summe groß genug ist, d.h., wenn die Ausführungskosten für genügend viele Aktivitäten aufaddiert werden. Um ein Gefühl für die Größenordnung der dafür notwendigen Anzahl zu bekommen, wollen wir folgendes Zahlenbeispiel betrachten:

$In\_parameter\_size_n$	= 300 kB
$Out\_parameter\_size_n$	= 100 kB
$Migration\_size_{m,n}$	= 1000 kB
$WL\_insert\_size_n$	= 0,2 kB
$WL\_delete\_size_n$	= 0,2 kB
$\#User_n(1)$	= 10
$\#User_n(2)$	= 200 $\Rightarrow$ $\#User_n = 210$

Bei diesen Zahlenwerten lohnt sich eine Migration, falls:

$$2 \cdot 1000 \text{ kB} < (200 - 10) \cdot |N_2| \cdot \left(\frac{300 \text{ kB} + 100 \text{ kB}}{210} + (0,2 \text{ kB} + 0,2 \text{ kB})\right)$$

$$\Leftrightarrow |N_2| > 4,57$$

Das bedeutet, dass sich in dem gegebenen Beispiel eine Migration lohnt, wenn die Zielpartition N_2 mindestens 5 Aktivitäten enthält. Bei der soeben durchgeführten Berechnung wurden keine an Gateways anfallenden Kosten berücksichtigt. Für diese ist eine analoge Betrachtung möglich, die zu einem Ergebnis in derselben Größenordnung führt.

4.8 Zusammenfassung und Ausblick

In verteilten Betriebssystemen und bei Mehrprozessormaschinen ist es üblich, dass die Verarbeitungseinheit für einen Prozess automatisch ausgewählt wird. Da i.d.R. kaum Information über die Prozesse vorhanden ist, wird dabei lediglich versucht, die Last der Prozessoren zu balancieren. Es ist eher unüblich, dass für einen Prozess ein aufgrund seiner Eigenschaften optimaler Prozessor ausgewählt wird. Dies ist aber bei WfMS notwendig, um eine bezüglich der Kommunikationskosten effiziente WF-Steuerung zu realisieren. Voraussetzungen dafür sind, dass die Struktur und Eigenschaften der WF-Typen bekannt sind, und dass ein Kostenmodell existiert, mit dem konkrete Verteilungen bewertet werden können. Wir haben diesen in der WF-Forschung neuen Weg eingeschlagen und ein Kostenmodell entwickelt, das es erlaubt, bei gegebenen Serverzuordnungen, WF-Modell und einigen Zusatzdaten, die bei der Ausführung von WF-Instanzen entstehenden Kosten abzuschätzen. Um geeignete Serverzuordnungen automatisch berechnen zu können, haben wir außerdem entsprechende Verteilungsalgorithmen entwickelt. Dabei handelt es sich sowohl um Verfahren, die eine optimale Verteilung ermitteln, als auch um solche, die eine gute Näherung berechnen. Außerdem wurden Ideen für weitere Verfahren skizziert. Ferner haben wir in diesem Kapitel einen Algorithmus zur Berechnung von Clustern von Benutzern vorgestellt, die aufgrund ihrer Aufgabenstruktur demselben Domain angehören sollten. Schließlich wurde gezeigt, dass der gewählte Ansatz zu einem skalierbaren WfMS führt, und dass Migrationen dazu beitragen können, die Belastung der Teilnetze zu reduzieren. Ein Vergleich mit anderen Verteilungsmodellen wird in Kapitel 9 durchgeführt, nachdem das Verteilungsmodell von ADEPT vollständig beschrieben wurde.

Das vorgestellte Kostenmodell wird primär benötigt, um geeignete Serverzuordnungen berechnen zu können. Es erlaubt aber auch Optimierungen in andere Richtungen. So haben z.B. Änderungen an den Bearbeiterzuordnungen der Aktivitäten oder an der Struktur der WF-Vorlagen Einfluss auf die Belastung der Teilnetze. Es gibt Szenarien, bei denen in solchen Punkten Freiheitsgrade existieren (z.B. weil die Netzlast ein so großes Problem ist, dass sogar Änderungen der betrieblichen Abläufe in Erwägung gezogen werden). In solchen Fällen kann das Kostenmodell verwendet werden, um den durch eine mögliche Veränderung erzielbaren Gewinn abzuschätzen. Das Kostenmodell ermöglicht also auch eine Optimierung des Organisationsmodells und der WF-Vorlagen. Ähnlich wie bei der Optimierung des Ortes für externe Datenquellen (vgl. Abschnitt 4.3.3) wird bei der Optimierung des Organisationsmodells vom WfMS kein Vorschlag berechnet. Stattdessen ermöglicht das Kostenmodell, verschiedene Alternativen zu vergleichen, um dann die beste auswählen zu können.

Kapitel 5

Variable Serverzuordnungen

Die in Kapitel 3 und 4 vorgestellten statischen Serverzuordnungen führen zu guten Ergebnissen, wenn unabhängige Bearbeiterzuordnungen (vgl. Abschnitt 2.4.2, z.B. `Rolle = Arzt`) verwendet werden. In diesem Fall steht die Menge der potentiellen Bearbeiter einer Aktivität schon zu Modellierungszeit fest, so dass ein WF-Server gewählt werden kann, der sich nahe bei diesen Bearbeitern befindet. Im vorliegenden Kapitel wird untersucht, welche Auswirkungen es auf die Verteilung hat, wenn in einem WF viele abhängige Bearbeiterzuordnungen verwendet werden. Ein Beispiel für einen WF mit abhängigen Bearbeiterzuordnungen ist in Abb. 5.1 dargestellt: Derselbe Arzt, der einen Patienten untersucht hat (Aktivität m), soll auch den zugehörigen Arztbrief schreiben (Nachfolgeraktivität n'). Außerdem soll der Patient von einer Pflegekraft derjenigen OE versorgt werden (Aktivität n), auf der er zuvor untersucht wurde. In diesem Beispiel steht erst nachdem die Aktivität m ausgeführt wurde, die OE der Bearbeiter der nachfolgenden Aktivitäten (n und n') fest. Wenn statische Serverzuordnungen verwendet werden, kann dieses Wissen aber nicht genutzt werden, da es erst bei der Ausführung der WF-Instanz entsteht. Genau dies sollte aber bei den Aktivitäten n und n' geschehen, um den WF-Server geeignet auszuwählen. Dies lässt sich durch die Verwendung von variablen Serverzuordnungen erreichen, da es diese auf effiziente Art und Weise erlauben, Laufzeitinformation der WF-Instanz zu berücksichtigen. Das entsprechende Konzept wurde im Rahmen der vorliegenden Arbeit entwickelt (siehe auch [BD99a, BD00a]). Des Weiteren wurden Verfahren entwickelt, die es erlauben, die variablen Serverzuordnungen weitgehend automatisch zu ermitteln. Schließlich sei noch bemerkt, dass es durch variable Serverzuordnungen zur Ausführungszeit einer WF-Instanz möglich ist, auf äußerst effiziente Weise die optimalen WF-Server für die Aktivitäteninstanzen zu bestimmen.

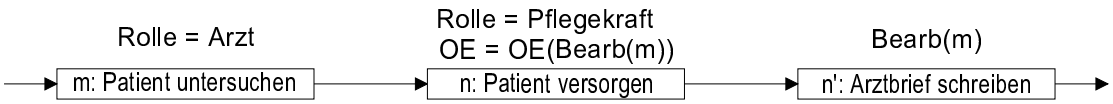


Abbildung 5.1 Beispiel für einen WF, der abhängige Bearbeiterzuordnungen enthält.

In Abschnitt 5.1 werden das Konzept der variablen Serverzuordnungen eingeführt und dessen Anwendung demonstriert. Zuvor werden mögliche alternative Vorgehensweisen untersucht. Auswirkungen variabler Serverzuordnungen, die zur Ausführungszeit der WF-Instanzen auftreten, werden in Abschnitt 5.2 diskutiert. Den Hauptteil dieses Kapitels bildet die Vorstellung von Verfahren zur automatischen Berechnung der vom Kostenmodell benötigten Wahrscheinlichkeitsverteilungen (WV). Dies sind sowohl die WV, die zur Berechnung der bei der Aktivitätenausführung entstehenden Kosten benötigt werden (Abschnitt 5.3), als auch diejenigen, die zur Berechnung der bei Migrationen

auftretenden Kosten benötigten werden (Abschnitt 5.4). Das Kapitel endet in Abschnitt 5.5 mit einer Zusammenfassung und einem Ausblick auf weitere Fragestellungen.

5.1 Grundlagen variabler Serverzuordnungen

Dieser Abschnitt schafft die Grundlagen für die weitere Betrachtung von variablen Serverzuordnungen. Dazu werden alternative Vorgehensweisen diskutiert und die bei variablen Serverzuordnungen verwendeten Ausdrücke eingeführt. Dann werden einige Anwendungsfälle aufgezeigt, in denen dieses Konzept große Vorteile mit sich bringt. Außerdem wird untersucht, welche Aktivitäten in einer bestimmten variablen Serverzuordnung referenziert werden dürfen. Schließlich wird noch beschrieben, wie die in Kapitel 4.5 vorgestellten Verteilungsalgorithmen für die Berechnung von variablen Serverzuordnungen verwendet werden können.

5.1.1 Verfahren zur Festlegung eines Workflow-Servers

Das Ziel dieses Abschnittes ist es, verschiedene Verfahren zur Festlegung eines WF-Servers zu vergleichen und ein geeignetes Verfahren auszuwählen. Dabei wird angenommen, dass der WF abhängige Bearbeiterzuordnungen enthält. Wir betrachten im Folgenden die Verwendung statischer Serverzuordnungen, die (voll dynamische) Festlegung des Servers einer Aktivität zur Ausführungszeit der WF-Instanz und die Verwendung variabler Serverzuordnungen.

5.1.1.1 Verwendung statischer Serverzuordnungen

Die einfachste und in der Literatur am häufigsten verwendete (vgl. Kapitel 9) Methode zur Festlegung von WF-Servern ist, jeder Aktivität zur Modellierungszeit oder beim Start der entsprechenden WF-Instanz fest (statisch) einen Server zuzuordnen. Diese Vorgehensweise kann auch dann verwendet werden, wenn in einem Prozess zahlreiche abhängige Bearbeiterzuordnungen vorkommen: Die in Abschnitt 4.5 vorgestellten Algorithmen zur Berechnung von geeigneten statischen Serverzuordnungen betrachten die Menge der potentiellen Bearbeiter $PotBearb_n$ einer Aktivität n. Diese Menge kann mit Algorithmus 2.1 auch für abhängige Bearbeiterzuordnungen berechnet werden. Allerdings qualifizieren sich bei abhängigen Bearbeiterzuordnungen nicht alle Benutzer aus $PotBearb_n$ tatsächlich bei jeder WF-Instanz für die Bearbeitung der Aktivität n. Angenommen, im Beispiel aus Abb. 5.1 gibt es 2 OE mit jeweils 5 Pflegekräften. Dann sind alle 10 Pflegekräfte in der Menge $PotBearb_n$ enthalten, weil sie alle für die Bearbeitung der Aktivität n in Frage kommen. Bei der Ausführung einer WF-Instanz qualifizieren sich für die Bearbeitung der Aktivität n aber nur die 5 Pflegekräfte der OE desjenigen Arztes, welcher die Aktivität m ausgeführt hat.

$PotBearb_n$ stellt also eine Obermenge der sich tatsächlich für eine Instanz der Aktivität n qualifizierenden Bearbeiter dar. Wird $PotBearb_n$ verwendet, um statische Serverzuordnungen zu berechnen, so ist aufgrund dieser Ungenauigkeit die Qualität der ermittelten Serverzuordnungen äußerst zweifelhaft. Dies gilt insbesondere, da die Beziehung zwischen Aktivitäten und OE verloren geht: In $PotBearb_n$ sind die Bearbeiter aller OE enthalten, unabhängig davon, in welcher OE die WF-Instanz tatsächlich bearbeitet wird. Bei Verwendung statischer Serverzuordnungen wird eine Aktivität bei allen WF-Instanzen des entsprechenden Typs vom selben Server kontrolliert. Für zahlreiche WF-Instanzen ist dessen Domain jedoch ungeeignet. Dies ist besonders nachteilig für WF-Typen, die

für die Dauer sehr vieler Aktivitäten in derselben (a priori unbekannten) OE verbleiben. In diesem Fall könnten, durch eine einzige Migration in den entsprechenden Domain, sehr viele Aktivitäten von dem günstigeren Server kontrolliert werden, wodurch wesentlich geringere Kosten entstehen würden. Aus diesem Grund ist die Verwendung von statischen Serverzuordnungen bei WF-Vorlagen mit vielen abhängigen Bearbeiterzuordnungen ungünstig.

5.1.1.2 Dynamische Festlegung des Workflow-Servers

Zur Ausführungszeit der WF-Instanzen stehen Laufzeitdaten (wie z.B. der tatsächliche Bearbeiter der Aktivität m aus Abb. 5.1) zur Verfügung. Prinzipiell kann man die besten Serverzuordnungen erhalten, wenn jeweils nach Beendigung einer Aktivität die WF-Server für die aktivierbaren Nachfolgeraktivitäten festgelegt werden. Zur Auswahl eines WF-Servers stehen dann die kompletten und aktuellen Laufzeitdaten der WF-Instanz zur Verfügung, so dass der tatsächlich am besten geeignete WF-Server bestimmt werden kann.

Diese Vorgehensweise bedeutet einen großen Aufwand für die WF-Server. Diese ohnehin schon stark belasteten Server müssen zusätzlich komplexe Analysen zur Festlegung der WF-Server durchführen. Es genügt nämlich nicht, den nur bzgl. der Nachfolgeraktivität n optimalen Server auszuwählen. Dies würde (analog zu den in Phase 1 der Verteilungsalgorithmen 4.3 - 4.5 berechneten Serverzuordnungen) zu sehr vielen nicht rentablen Migrationen führen. Deshalb muss zur Festlegung des WF-Servers einer zu aktivierenden Aktivität n der gesamte restliche WF analysiert werden.

Das Verfahren kann vereinfacht werden, indem nur die nächsten K Aktivitäten bei der Analyse betrachtet werden (z.B. mit $K = 10$). Aber auch dann können sehr komplexe Wahrscheinlichkeitsabschätzungen notwendig werden, wie die Beispiele aus Abb. 5.2 zeigen. Bei diesen Beispielen wurde jeweils die Aktivität m soeben beendet, so dass der Server für die Nachfolgeraktivität n festgelegt werden muss. In Abb. 5.2a hängt die OE der Nachfolgeraktivitäten n', n'' ... der Aktivität n vom Bearbeiter von n ab. Dieser ist aber noch nicht bekannt. Lediglich die potentiellen Bearbeiter von Aktivität n können ermittelt werden. Für jeden dieser potentiellen Bearbeiter werden nun Wahrscheinlichkeitsbetrachtungen für die Aktivitäten n', n'' ... nötig, um den Server der Aktivität n so festlegen zu können, dass im weiteren Ablauf möglichst wenig Migrationen auftreten. Dabei kommt erschwerend hinzu, dass die Bearbeiter der Aktivität n zu mehreren Domains gehören können. Deshalb ist es schon für die Aktivität n isoliert betrachtet im Voraus nicht immer möglich, den zugehörigen WF-Server zu bestimmen. In Abb. 5.2b stellt die Aktivität n einen OR-Split-Knoten dar. Deshalb muss bei der Betrachtung der Nachfolgeraktivitäten von n die Wahrscheinlichkeit berücksichtigt werden, mit welcher der jeweilige Zweig gewählt wird. Da, wie für die Aktivität n'' angedeutet, Verzweigungen auch verschachtelt werden können, führt dies zu aufwendigen Wahrscheinlichkeitsbetrachtungen für die Nachfolgeraktivitäten von m. In Abb. 5.2c ist der Nachfolger von Aktivität n der AND-Join-Knoten n'. Es kann günstig sein, diese Aktivität vom selben Server kontrollieren zu lassen, wie deren Vorgängeraktivität m', um zwischen diesen eine Migration zu vermeiden. Dieser Server s_2 ist dem Server s_1, der die Aktivität n kontrolliert, aber nicht bekannt – eventuell steht s_2 noch gar nicht fest, weil die Ausführung des entsprechenden Zweiges noch nicht so weit fortgeschritten ist. Deshalb kann nicht entschieden werden, ob die Aktivität n ebenfalls von diesem Server kontrolliert werden soll, um eine Migration zwischen n und n' zu vermeiden. Es sind lediglich wieder komplexe Wahrscheinlichkeitsbetrachtungen möglich, um abzuschätzen, welcher Server für die Aktivität n am besten geeignet ist.

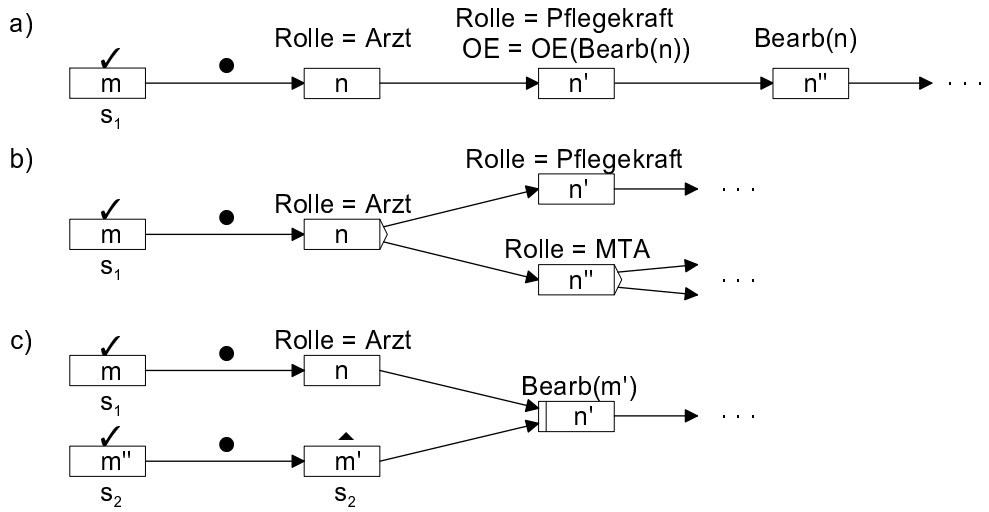


Abbildung 5.2 Fälle, in denen die dynamische Festlegung des WF-Servers für die Aktivität n komplexe Analysen erfordert.

Aufgrund der beschriebenen Schwierigkeiten und da auch noch Kombinationen dieser und weiterer Problemfälle auftreten können (z.B. bei Schleifen), ist der für die WF-Server entstehende Aufwand bei diesem Verfahren sehr groß. Da sie diesen Aufwand in einem Produktions-WfMS nicht zusätzlich bewältigen können, scheidet es i.d.R. aus. Außerdem werden bei einem solchen Verfahren keine wesentlich besseren Ergebnisse erzielt, als bei Verwendung variabler Serverzuordnungsausdrücke: Weil aus Aufwandsgründen bei der Analyse nur die nächsten K Aktivitäten betrachtet werden können und Vereinfachungen bei der Analyse notwendig sind, weichen die berechneten Serverzuordnungen u.U. deutlich vom Optimum ab, so dass mit vorberechneten variablen Serverzuordnungen sogar eine bessere Verteilung erreicht werden kann.

5.1.1.3 Verwendung variabler Serverzuordnungen

Variable Serverzuordnungen stellen eine Kombination von statischer Vorberechnung (zur Modellierungszeit) mit dynamischen Aspekten (Berücksichtigung von Laufzeitdaten) dar. Damit werden die wesentlichen Vorteile der beiden Vorgehensweisen vereint. Die Funktionsweise von variablen Serverzuordnungen soll an dem in Abb. 5.1 dargestellten Beispiel erläutert werden: Wegen den abhängigen Bearbeiterzuordnungen steht für die Aktivitäten n und n' erst (unmittelbar) vor ihrer Ausführung fest, aus welcher OE die Bearbeiter stammen werden. Durch den Einsatz von variablen Serverzuordnungen ist es nun möglich, stets den optimalen WF-Server für diese Aktivitäten zu verwenden. Anstelle einer statischen Serverzuordnung der Art `Aktivität m wird vom Server 5 kontrolliert`, werden logische Serverzuordnungsausdrücke verwendet, z.B. `Aktivität n wird vom Server im Domain desjenigen Bearbeiters kontrolliert, der Aktivität m ausgeführt hat`. Es wird also Information genutzt, die zur Modellierungszeit bzw. beim Starten der WF-Instanz noch nicht existiert hat. Dadurch wird erreicht, dass stets (d.h. bei allen WF-Instanzen) der optimale WF-Server gewählt wird.

Zur Modellierungszeit werden also anstelle von festen Serverzuordnungen (wo erforderlich bzw. sinnvoll) logische Serverzuordnungsausdrücke festgelegt, die ebenfalls weitgehend automatisch berechnet werden können. Diese referenzieren Laufzeitdaten der WF-Instanz und ermöglichen so auf einfache

Weise eine Festlegung des konkreten WF-Servers zur Ausführungszeit. Diese Festlegung ist zudem äußerst effizient, da zur Auswertung einer variablen Serverzuordnung lediglich ein sehr einfacher Ausdruck (vgl. Abschnitt 5.1.2) ausgewertet werden muss. Dadurch wird der Aufwand, wie bei statischen Serverzuordnungen, zur Modellierungszeit betrieben. Deshalb werden die WF-Server kaum zusätzlich belastet und es ist stets möglich, sehr gut geeignete Server festzulegen. Aus diesen Gründen wird dieses Verfahren weiter verfolgt.

5.1.2 Definition möglicher Serverzuordnungsausdrücke

In diesem Abschnitt werden die Syntax und die Semantik von variablen Serverzuordnungen festgelegt. In den nachfolgenden Abschnitten wird dann untersucht, für welche Szenarien solche Serverzuordnungen geeignet sind, und wie geeignete (variable) Serverzuordnungen für die Aktivitäten einer WF-Vorlage ermittelt werden können.

Bei statischen Serverzuordnungen werden als Zuordnungsausdrücke die IDs der WF-Server verwendet. Im Fall variabler Serverzuordnungen sind, wie oben bereits erwähnt, als Serverzuordnungen auch Ausdrücke erlaubt, die Laufzeitdaten der WF-Instanz referenzieren. In den meisten praktisch relevanten Fällen wird man sich hierbei auf den Server oder den Bearbeiter einer Vorgängeraktivität beschränken. In Sonderfällen können auch weitere WF-interne Daten (z.B. der Startzeitpunkt einer Aktivität), beliebige mathematische Funktionen, logische Ausdrücke oder die Einbeziehung WF-relevanter Daten (Parameterdaten von Aktivitäten) Sinn machen. Während Serverzuordnungen der erstgenannten Art (s.u.: 1. - 5.) systemseitig vom WF-Modell abgeleitet werden können, müssen die Zuordnungsausdrücke für die Sonderfälle (6.) explizit vom Modellierer vorgegeben werden.

In Folgenden werden die möglichen Serverzuordnungsausdrücke festgelegt und deren Semantik definiert. Dazu wird angegeben, welche Bedingung der tatsächlich zur Kontrolle der Aktivitäteninstanz n ausgewählte WF-Server $Server_n$ (siehe Definition 3.2) erfüllen muss.[1] Die Auswertung eines Serverzuordnungsausdrucks $ServZuordn_n$ findet immer dann statt, wenn eine entsprechende Aktivitäteninstanz aktiviert werden soll. Dies kann für eine Aktivität, die innerhalb einer Schleife liegt, auch mehrfach geschehen.

1. $ServZuordn_n =$ `s`
   Der Aktivität n wird der Server s fest (statisch) zugeordnet:
   $Server_n = s$
2. $ServZuordn_n =$ `Server(m)`
   Die Aktivität n soll vom selben Server kontrolliert werden wie die Aktivität m:
   $Server_n = \begin{cases} \text{undefiniert} & \text{, falls } Server_m = \text{undefiniert} \\ Server_m & \text{, sonst} \end{cases}$
3. $ServZuordn_n =$ `Domain(Bearb(m))`
   Der Aktivität n wird der Server zugewiesen, der sich im Domain des Bearbeiters befindet, der die Aktivität m ausgeführt hat:
   $Server_n =$ undefiniert , falls der Bearbeiter von Aktivität *m* noch nicht feststeht
   $Domain(Server_n) = Domain(u)$ mit $u = Bearbeiter(m)$, sonst
4. $ServZuordn_n =$ `f(Server(m))`
   Auf eine Serverzuordnung vom Typ 2 kann noch eine Funktion $f : S \mapsto S$ (S ist die Menge der

[1] An dieser Stelle wird auch der Sinn der Unterscheidung zwischen dem Serverzuordnungsausdruck $ServZuordn_n$ und dem tatsächlichen WF-Server $Server_n$ der Aktivität n klar.

WF-Server) angewandt werden:

$$Server_n = \begin{cases} \text{undefiniert} & \text{, falls } Server_m = \text{undefiniert} \\ f(Server_m) \text{ wobei } f \text{ eine vorgegebene Funktion ist} & \text{, sonst} \end{cases}$$

Da die WV der beiden Aktivitäten m und n berechnet werden können (siehe Abschnitt 5.3), kann eine optimale Funktion f automatisch erzeugt werden. Wie dies funktioniert, wird im Abschnitt 5.3.3.2 beschrieben.

5. $ServZuordn_n =$ `f(Domain(Bearb(m)))`
   Auch auf Serverzuordnungen vom Typ 3 kann eine Funktion f angewandt werden:
   $Server_n =$ undefiniert , falls der Bearbeiter von Aktivität *m* noch nicht feststeht
   $Domain(Server_n) = f(Domain(u))$ mit $u = Bearbeiter(m)$
   wobei *f* eine vorgegebene Funktion ist , sonst

6. $ServZuordn_n = Ausdruck$
   Der Modellierer kann auch einen beliebigen Serverzuordnungsausdruck $Ausdruck$ vorgeben, der nicht Typ 1 - 5 entspricht. Dieser wird dann bei der Ausführung der Aktivität n ausgewertet:

$$Server_n = \begin{cases} \text{undefiniert} & \text{, falls } \textit{Ausdruck} \text{ noch nicht ausgewertet werden kann} \\ val(Ausdruck) & \text{, sonst} \end{cases}$$

   Ein solcher Ausdruck kann z.B. in Form eines einfachen Programms, mit eingeschränktem Befehlssatz, vorgegeben werden. Da dieses vom WfMS nicht analysiert werden kann, können die WV nicht automatisch berechnet werden. Deshalb muss der Modellierer auch diese vorgeben, weil sie zur Abschätzung der entstehenden Kosten benötigt werden.

Wenn variable Serverzuordnungen verwendet werden, dann kann in $ServZuordn_n$ eine Aktivität m referenziert werden. Um diese beschreiben zu können, wird in Definition 5.1 die Funktion $ReferencedAct$ eingeführt:

Definition 5.1 (in Serverzuordnung referenzierte Aktivität: $ReferencedAct$)
N sei die Menge der Aktivitäten einer WF-Vorlage. Dann definiert die Funktion *ReferencedAct* für eine Aktivität $n \in N$ die in deren Serverzuordnung referenzierte Aktivität:
$ReferencedAct : N \mapsto N \cup \{NULL\}$ mit

$$ReferencedAct_n = \begin{cases} NULL & \text{, falls } ServZuordn_n \text{ vom Typ 1 oder 6} \\ m & \text{, falls } ServZuordn_n \text{ von der Bauart 2 - 5 (s.o.)} \end{cases}$$

Zur Ausführungszeit einer WF-Instanz muss vor dem Aktivieren der Aktivität n der Ausdruck $ServZuordn_n$ ausgewertet werden. Damit dies immer möglich ist, muss sichergestellt sein, dass die in $ServZuordn_n$ referenzierte Aktivität $m = ReferencedAct_n$ garantiert vor der Aktivität n ausgeführt wird. Das bedeutet, dass die Aktivität m immer vor der Aktivität n ausgeführt werden muss. Außerdem muss, falls die Aktivität n tatsächlich ausgeführt wird, auch die Aktivität m wirklich ausgeführt worden sein, d.h. sie darf sich dann nicht in einem nicht gewählten Zweig einer bedingten Verzweigung befinden. Nur dann sind die benötigen Laufzeitdaten (Bearbeiter, Server) verfügbar. Wie diese Bedingungen gewährleistet werden können, wird im Abschnitt 5.1.4.1 diskutiert.

5.1.3 Anwendungen für variable Serverzuordnungen

Im Beispiel zur Abb. 5.1 haben wir schon gesehen, wie variable Serverzuordnungen eingesetzt werden können. In diesem Abschnitt wird der Nutzen von variablen Serverzuordnungen an weiteren Beispie-

len demonstriert. Diese Beispiele können auch kombiniert werden, so dass sich weitere Anwendungen für variable Serverzuordnungen ergeben.

In dem im Abb. 5.3a dargestellten Beispiel werden die Aktivitäten $b - n$ vom selben Benutzer u bearbeitet, wie die Aktivität a. Für diese Aktivitäten wird die Serverzuordnung `Domain(Bearb(a))` verwendet. Dadurch wird derjenige WF-Server gewählt, der sich im Domain dieses Benutzers u befindet. Da der Benutzer u auch diese Aktivitäten bearbeitet, ist ihre Ausführung bei dieser Serverzuordnung auf ein Teilnetz beschränkt, d.h. kein Gateway (und nur ein einziges Teilnetz) wird mit den entsprechenden Kommunikationen belastet.

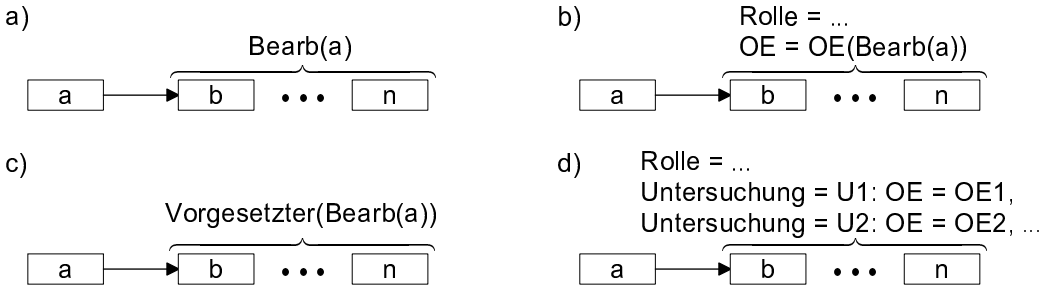


Abbildung 5.3 Beispiele für die Verwendung von variablen Serverzuordnungen.

In Abb. 5.3b stammen die Bearbeiter der Aktivitäten $b - n$ aus derselben OE wie der Bearbeiter der Aktivität a. Befinden sich alle Bearbeiter einer OE im selben Domain, so ist die Ausführung der Aktivitäten $b - n$ durch die Serverzuordnung `Domain(Bearb(a))` wieder auf ein Teilnetz beschränkt. Selbst wenn sich OE und Domains nicht exakt entsprechen, so ist doch anzunehmen, dass sich die meisten Benutzer einer OE im selben Teilnetz oder zumindest in benachbarten Teilnetzen befinden. Dadurch ist die Kommunikation zwischen dem WF-Server und den meisten potentiellen Bearbeitern der Aktivitäten auf ein Teilnetz beschränkt. Zumindest wird die Anzahl der weit entfernten Kommunikation stark reduziert.

In Abb. 5.3c ist der Fall dargestellt, dass Aktivitäten vom Vorgesetzten des Bearbeiters der Aktivität a ausgeführt werden. Dies führt zum selben Ergebnis wie im vorherigen Beispiel, wenn die Bearbeiter von Aktivität a und ihr Vorgesetzter jeweils derselben OE und demselben Domain angehören (zumindest für den Großteil der Paare Mitarbeiter – Vorgesetzter). Es gibt aber auch Szenarien, in denen die Vorgesetzten einer anderen (Management-) OE und einem anderen Domain angehören. In diesen Fällen sind die im vorherigen Beispiel verwendeten Serverzuordnungen ungeeignet, da die Aktivitäten $b - n$ vom WF-Server im Domain des Vorgesetzten und nicht vom Server im Domain der Untergebenen kontrolliert werden sollen. Stattdessen müssen Serverzuordnungen der Art `f(Domain(Bearb(a)))` verwendet werden. Die Funktion f bildet dabei vom Domain der Untergebenen einer OE zum Domain ihres Vorgesetzten ab. Da für die Benutzer aus einer bestimmten OE schon zur Modellierungszeit feststeht, welchem Domain ihr Vorgesetzter angehört, wird diese Funktion f schon zur Modellierungszeit festgelegt.

Eine weitere Anwendung für die Serverzuordnung `f(Domain(Bearb(a)))` ergibt sich bei Bearbeiterzuordnungen der Art `Rolle = B` $\wedge$ `OE = OE(Bearb(a))` (vgl. Abb. 5.3b). Angenommen, der Bearbeiter von Aktivität a habe die Rolle A. Wenn Benutzer derselben OE, aber mit unterschiedlichen Rollen, in verschiedenen Domains angesiedelt sind, so muss die Funktion f vom Domain der Benutzer einer OE mit Rolle A zu den Benutzern dieser OE mit Rolle B abbilden.

Ergibt sich die OE der Bearbeiter der Aktivitäten $b-n$ aus Datenelementen des WF (siehe Abb. 5.3d), so muss auch der entsprechende WF-Server aus diesen Daten ermittelt werden. Dazu können vom

Modellierer Serverzuordnungsausdrücke vom Typ 6 vorgegeben werden. Im Beispiel aus Abb. 5.3d eignet sich für die Aktivitäten $b - n$ die Serverzuordnung `Untersuchung = U1:` s_{OE1}`, Untersuchung = U2:` s_{OE2}.

Ein letzter, aber besonders wichtiger, Anwendungsbereich für variable Serverzuordnungen ergibt sich bei der Verwendung von *Standard-WF-Typen.* Damit meinen wir Prozesse, die einmal entworfen und dann in unterschiedlichen OE verwendet werden (z.B. der Kreditantrags-WF aus Abschnitt 1.2.2). Ein Standard-WF verbleibt meist in derjenigen OE, in welcher er gestartet wurde. Bei statischer Serverzuordnung befinden sich die WF-Server i.d.R. nicht in dieser OE. In ADEPT$_{distribution}$ wird der in Abschnitt 2.2.1 erwähnte explizite Startknoten $Start$ eines WF (dem keine Aktivität zugeordnet ist) stets vom WF-Server im Domain desjenigen Benutzers kontrolliert, welcher den WF gestartet hat. Bei Standard-WF ist es dann günstig, wenn alle Aktivitäten die Serverzuordnung `Server(Start)` verwenden, so dass der WF in dem Domain (und damit der OE) verbleibt, in welchem er gestartet wurde. Dadurch sind keine Migrationen notwendig und alle Kommunikationen sind auf diese OE beschränkt.

5.1.4 Referenzierbare Aktivitäten

Nachdem geklärt wurde, welche Ausdrücke als $ServZuordn_n$ in Frage kommen und wie variable Serverzuordnungen eingesetzt werden können, wird in diesem Abschnitt diskutiert, welche Aktivitäten m durch einen bestimmten Ausdruck referenziert werden sollen. Dazu wird untersucht, welche Aktivitäten in einer Serverzuordnung überhaupt referenziert werden dürfen. Dann wird analysiert, bei welchen Aktivitäten es Sinn macht, sie in einer bestimmten Serverzuordnung zu referenzieren.

5.1.4.1 Für eine Referenzierung zulässige Aktivitäten

Damit die Aktivität m in $ServZuordn_n$ referenziert werden darf, muss sie garantiert vor der Aktivität m ausgeführt werden (vgl. Abschnitt 5.1.2). Dazu muss die Aktivität $m = ReferencedAct_n$ im Ablaufgraphen vor n liegen und die Aktivität m muss stets ausgeführt werden, falls n ausgeführt wird. Um zu zeigen, dass auch die zweite Bedingung notwendig ist, ist in Abb. 5.4 ein Beispiel dargestellt, bei dem nur diese verletzt ist: In $ServZuordn_n$ darf die Aktivität m' nicht referenziert werden, obwohl m' eine Vorgängeraktivität von n ist ($m' \in Pred^*_{\{CONTROL\_E,SYNC\_E\}}(n)$), weil nicht sichergestellt ist, dass m' überhaupt ausgeführt wird.

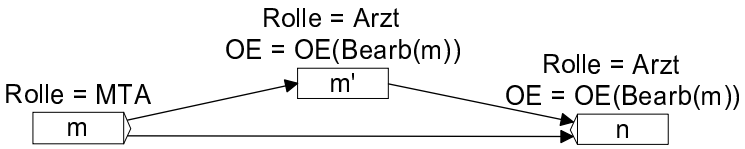


Abbildung 5.4 Die Vorgängeraktivität m' darf nicht in der $ServZuordn_n$ referenziert werden.

Um sicherzustellen, dass die Aktivität m stets ausgeführt wird, falls die Aktivität n ausgeführt wird, prüft die in Algorithmus 5.1 definierte Funktion $RefAllowed$, ob sich m in einem Zweig einer bedingten Verzweigung befindet, in dem sich n nicht befindet.[2] Ist m eine Vorgängeraktivität von n bezüglich normaler Kontrollkanten, so befindet sie sich (wegen der Blockstrukturierung von ADEPT,

[2]Die Blockstrukturierung erweist sich als sehr hilfreich bei der Durchführung der entsprechenden Analysen. Bei allgemeinen (z.B. Petri-Netz-basierten) WF-Modellen sind die zur Lösung dieser Fragestellung notwendigen Algorithmen wesentlich komplexer und aufwendiger.

vgl. Abschnitt 2.2.1) in einer solchen bedingten Verzweigung, wenn sich zwischen m und n ein Verzweigungs-Endknoten und nicht der zugehörige Verzweigungs-Anfangsknoten befindet. Ob dies gegeben ist, wird durch die Funktion $notInConditionalBranch$ geprüft. Dazu betrachtet die Funktion alle OR-Join-Knoten j, die sich zwischen den Aktivitäten n (einschließlich) und m befinden. Falls der zu der Aktivität j gehörende OR-Split-Knoten s ein Vorgänger der Aktivität m ist, befindet sich m innerhalb dieser bedingten Verzweigung, so dass False zurückgegeben wird. Dann darf die Aktivität m nicht referenziert werden.

Ist die Aktivität m nur dann ein Vorgänger von n, wenn auch Synchronisationskanten berücksichtigt werden, so befindet sich m in einem zu n parallel ausgeführten Zweig. In diesem Fall liegt eine bedingte Verzweigung, welcher die Aktivität m angehören könnte, ebenfalls in diesem parallelen Zweig. Um zu überprüfen, ob die Aktivität m stets ausgeführt wird, falls n ausgeführt wird, ermittelt der Algorithmus den AND-Join-Knoten p der parallelen Zweige von m und n. Da die Aktivität p das Ende der Parallelität darstellt, wird p garantiert ausgeführt, falls n ausgeführt wird. Der Algorithmus prüft nun unter Verwendung der Funktion $notInConditionalBranch$, ob sich m in einer bedingten Verzweigung befindet, welcher p nicht angehört. Ist dies nicht der Fall, so wird m garantiert ausgeführt, falls p ausgeführt wird (und damit auch falls n ausgeführt wird), so dass das Funktionsergebnis True ist. Ist das Funktionsergebnis False, so darf die Aktivität m in $ServZuordn_n$ nicht referenziert werden. Dies ist auch dann der Fall, wenn m überhaupt keine Vorgängeraktivität von n ist (Else-Fall).

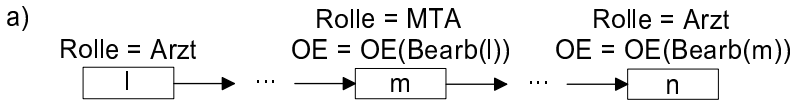
Algorithmus 5.1 (Prüfen, ob Referenzierung erlaubt: $RefAllowed$)
input
$CFS = (N, \ldots)$: Kontrollflussstruktur des WF-Typs
m: Aktivität, die referenziert werden soll
n: die referenzierende Aktivität
result
boolean: True genau dann, wenn die Aktivität m in $ServZuordn_n$ referenziert werden darf
begin
if $m \in Pred^*_{\{CONTROL\_E\}}(n)$ **then** // m ist Vorgängeraktivität von n bzgl. Kontrollkanten
return $notInConditionalBranch(CFS, n, m)$;
else if $m \in Pred^*_{\{CONTROL\_E, SYNC\_E\}}(n)$ **then** // bzgl. Kontroll- und Synchronisationskanten
$p = nextCommonAndJoin(CFS, n, m)$; // p ist Ende der Parallelität von m und n
return $notInConditionalBranch(CFS, p, m)$;
else return False; // m ist keine Vorgängeraktivität von n
end.

$notInConditionalBranch(CFS, n, m)$:
input
$CFS = (N, \ldots)$: Kontrollflussstruktur des WF-Typs
n, m: Aktivitäten mit der Eigenschaft $m \in Pred^*_{\{CONTROL\_E\}}(n)$
result
boolean: True, wenn sich m in keiner bedingten Verzeigung befindet, in der nicht auch n liegt
begin
for each $j \in N$ mit $V_{in}(j) = ONE\_Of\_ALL$ // j ist OR-Join-Knoten
$\wedge\ j \in Pred^*_{\{CONTROL\_E\}}(n) \cup \{n\}\ \wedge\ j \in Succ^*_{\{CONTROL\_E\}}(m)$ **do** // j liegt zwischen m und n
$s = Split(j)$; // s ist der zu j gehörende OR-Split-Knoten
if $s \in Pred^*_{\{CONTROL\_E\}}(m)$ **then return** False; // s liegt nicht zwischen m und n
return True;
end;

5.1.4.2 Für Referenzierung relevante Aktivitäten

Die oben erwähnte Funktion $RefAllowed$ prüft, ob es erlaubt ist, eine Aktivität m in $ServZuordn_n$ zu referenzieren. Daraus folgt noch nicht, dass dies auch sinnvoll ist. Deshalb wird in diesem Abschnitt untersucht, für welche Aktivitäten m es interessant ist, sie in $ServZuordn_n$ zu referenzieren.

Die nahe liegendste Idee ist, für $ServZuordn_n$ nur diejenige Aktivität m zu berücksichtigen, die in $BearbZuordn_n$ referenziert wird. Aber auch bzgl. der Bearbeiterzuordnungen transitiv abhängige Aktivitäten können relevant sein. In dem in Abb. 5.5a dargestellten Beispiel hat der Bearbeiter der (von Aktivität n transitiv abhängigen) Aktivität l dieselbe Rolle Arzt wie der Bearbeiter von Aktivität n. Außerdem gehört er derselben OE an. Der Bearbeiter der direkt abhängigen Aktivität m hat die Rolle MTA. Da mit der in Abb. 5.5c dargestellten Bearbeiterverteilung der Bearbeiter von Aktivität m immer im Domain 1 angesiedelt sind, ist diese Information zur Festlegung der Serverzuordnung von Aktivität n wertlos. Der Bearbeiter der Aktivität l (mit Rolle Arzt) befindet sich abhängig von seiner OE im Domain 1 oder 2. Da für den Bearbeiter der Aktivität n dasselbe gilt, ist der Domain des Bearbeiters der transitiv abhängigen Aktivität l die optimale Lokation für den Server von Aktivität n.



c) Anteil der Bearbeiter eines Domains (aufgeschlüsselt nach OE und Rolle):

MTA	Domain 1	Domain 2
OE 1	1	0
OE 2	1	0

Arzt	Domain 1	Domain 2
OE 1	1	0
OE 2	0	1

Abbildung 5.5 Die indirekt abhängige Aktivität l ist für $ServZuordn_n$ relevant.

Doch nur die transitiven Abhängigkeiten zu berücksichtigen, reicht nicht aus, wie Abb. 5.5b zeigt. Mit der Bearbeiterverteilung aus Abb. 5.5c befindet sich der Bearbeiter der einzigen von Aktivität n (transitiv) abhängigen Aktivität m immer im Domain 1 (wegen Rolle MTA). Der Bearbeiter der Aktivität l hat aber wie der Bearbeiter von Aktivität n die Rolle Arzt und er stammt auch aus derselben OE. Da er deshalb stets im gleichen Domain angesiedelt ist, ist dieser Domain der optimale Ort für den Server von Aktivität n. Daraus folgt, dass alle Aktivitäten für eine Referenzierung relevant sind, die indirekt (vor- oder rückwärts) mit Aktivität n in Beziehung stehen. Alle anderen Aktivitäten sind nicht relevant, da ihre Bearbeiter und deren OE in keiner Beziehung zum Bearbeiter von Aktivität n stehen.

Für die in Abschnitt 5.3 und 5.4 beschriebenen Analysen ist es notwendig, zu wissen, ob die Aktivität n und die in $ServZuordn_n$ referenzierte Aktivität m denselben Bearbeiter haben, bzw., ob deren Bearbeiter derselben OE angehören. Deshalb ermittelt der nachfolgend beschriebene Algorithmus 5.2 nicht nur die für eine Referenzierung relevanten Aktivitäten, sondern er ermittelt auch den Typ der Abhängigkeit zwischen den beiden Aktivitäten. Dabei haben die berechneten Werte folgende Bedeutung:

$Dep(n,m) = B$: Aktivität n hat denselben Bearbeiter wie Aktivität m
$Dep(n,m) = OE$: Aktivität n hat einen Bearbeiter aus derselben OE wie Aktivität m
$Dep(n,m) = NULL$: zwischen den Bearbeitern der Aktivitäten n und m besteht keine Abhängigkeit

Der Algorithmus 5.2 berechnet zuerst aus den Bearbeiterzuordnungen aller Aktivitäten initiale (nicht transitive) Abhängigkeiten Dep (Prozedur $initial\_dep$). Anschließend wird versucht, aus jedem Paar von Abhängigkeiten eine neue Abhängigkeit zu generieren. Diese Aufgabe übernimmt die Prozedur $deduce\_dep$, welche die in Abb. 5.5a und b dargestellten Fälle indirekter Abhängigkeiten berücksichtigt. Wurde ein Paar von Abhängigkeiten gefunden, das sich kombinieren lässt, so muss noch der Typ der neu erzeugten Abhängigkeit ermittelt werden (Funktion $combine$). Sind beide Abhängigkeiten vom Typ B (selber Bearbeiter), so hat auch die neue Abhängigkeit den Typ B. Sind sie vom Typ B oder OE, so kann für die neue Abhängigkeit nur dieselbe OE garantiert werden (Typ OE).

Der Algorithmus 5.2 analysiert Zusammenhänge zwischen den Bearbeiterzuordnungen der Aktivitäten, ist also von den Serverzuordnungen unabhängig. Er wird deshalb nicht vom Verteilungsalgorithmus (siehe Abschnitt 4.5) aufgerufen, sondern nur einmal (vor dem Berechnen der Serverzuordnungen) ausgeführt.

Wenn eine Abhängigkeit $Dep(n,m) = B$ oder $Dep(n,m) = OE$ existiert, so ist m für eine Referenzierung in $ServZuordn_n$ „interessant“. Damit die Aktivität m in $ServZuordn_n$ referenziert werden darf, muss sie außerdem garantiert vor Aktivität n ausgeführt werden ($RefAllowed(CFS, n, m) =$ True). Damit ergeben sich die in $ServZuordn_n$ referenzierbaren und relevanten Aktivitäten $RelevantAct_n$ wie folgt:
$RelevantAct_n = \{m \mid Dep(n,m) \neq NULL \ \wedge \ RefAllowed(CFS, n, m)\}$

5.1.4.3 Bearbeiter von Aktivitätenpaaren

Die Abhängigkeiten Dep wurden bestimmt, um entscheiden zu können, welche Aktivitäten in einer Serverzuordnung referenziert werden sollen. Die Abhängigkeiten beschreiben aber eigentlich (transitive) Beziehungen zwischen den Bearbeitern von zwei Aktivitäten. Deshalb können sie auch verwendet werden, um zu entscheiden, ob sich ein Benutzer für die Bearbeitung einer Aktivität qualifiziert, wenn der Bearbeiter einer anderen Aktivität vorgegeben ist. Da diese Funktionalität im Folgenden benötigt wird, wird in diesem Abschnitt ein entsprechender Algorithmus vorgestellt.

Die berechneten Abhängigkeiten Dep lassen erkennen, welches die potentiellen Bearbeiter einer Aktivität n sind, wenn der Bearbeiter einer anderen Aktivität m vorgegeben ist. Mit dieser Information lässt sich also berechnen, ob ein Bearbeiter u die Aktivität n bearbeiten darf, falls der Bearbeiter u' der Aktivität m gegeben ist. Dazu prüft der Algorithmus 5.3 zuerst, ob der Bearbeiter u – unabhängig vom Bearbeiter der Aktivität m – die Aktivität n bearbeiten darf, d.h. (vereinfacht), ob er z.B. die richtige Rolle hat. Dies ist gegeben, falls $u \in PotBearb_n$ gilt. Anschließend wird getestet, ob die Bearbeiter u' und u eine Abhängigkeit zwischen den Aktivitäten m und n verletzen. Existiert zwischen den Aktivitäten m und n eine Abhängigkeit vom Typ B (selber Bearbeiter), so darf u die Aktivität n nur bearbeiten, falls er mit u' identisch ist; besteht eine Abhängigkeit vom Typ OE, so müssen die Bearbeiter u und u' derselben OE angehören. Gibt es keine Abhängigkeit zwischen den Aktivitäten m und n, so darf der Bearbeiter u die Aktivität n immer bearbeiten, da dann auch keine Abhängigkeit verletzt sein kann. Dass der Bearbeiter u die Aktivität n aufgrund seiner Rolle o.Ä. prinzipiell bearbeiten darf, wurde schon am Anfang des Algorithmus geprüft.

Algorithmus 5.2 (Abhängigkeiten zwischen den Bearbeitern der Aktivitäten)
input
N: die Menge der Aktivitäten der WF-Vorlage
$BearbZuordn_n$: $\forall n \in N$: die Bearbeiterzuordnung der Aktivität n
output
$Dep(n,m)$: $\forall n,m \in N$: die Beziehung zwischen den Bearbeitern der Aktivitäten n und m
begin
$\forall n,m \in N$: $Dep(n,m) = NULL$;
for each Aktivität $n \in N$ **do** $initial\_dep(n)$; // die direkten Abhängigkeiten ermitteln
do // alle indirekten Abhängigkeiten ableiten
$changed$ = False;
for each $n,n' \in N$ mit $Dep(n,n') \neq NULL$ **do**
for each $m,m' \in N$ mit $Dep(m,m') \neq NULL$ **do**
// leite aus Abhängigkeit zwischen n, n' und m, m' neue Abhängigkeiten her
$Dep3 = deduce\_dep(n,n',m,m',l,l',T)$;
if $T \neq NULL \wedge Dep(l,l') = NULL$ **then** // neue Abhängigkeit gefunden
$Dep(l,l') = T$; $Dep(l',l) = T$; $changed$ = True;
while $changed$ = True;
end.

$initial\_dep(n)$:
input
n: die Aktivität, deren direkte Abhängigkeiten ermittelt werden sollen
begin
case $BearbZuordn_n$ = `Bearb(m)`:
$Dep(n,m) = B$; $Dep(m,n) = B$;
case $BearbZuordn_n$ = `OE` = `OE(Bearb(m))` $\wedge \ldots$:
$Dep(n,m) = OE$; $Dep(m,n) = OE$;
end;

$deduce\_dep(n,n',m,m',l,l',T)$:
input
n, n' und m, m': Paare von Aktivitäten mit existierenden Abhängigkeiten
output
l, l', T: abgeleitete Abhängigkeit vom Typ T zwischen l und l'
begin
if $n' = m$ **then** // Fall aus Abb. 5.5a
$T = combine(Dep(n,n'), Dep(m,m'))$; $l = n$; $l' = m'$;
else if $n = m$ **then** // Fall aus Abb. 5.5b
$T = combine(Dep(n,n'), Dep(m,m'))$; $l = n'$; $l' = m'$;
else
$T = NULL$;
end;

$combine(T_1, T_2)$:
input
T_1, T_2: Typen der zu kombinierenden Abhängigkeiten
result
Typ der daraus entstehenden Abhängigkeit
begin
if $T_1 = B \wedge T_2 = B$ **then return** B;
else return OE; // $T_1 \in$ {B, OE} und $T_2 \in$ {B, OE}
end;

Algorithmus 5.3 (Zulässigkeit eines Bearbeiterpaars: $ActorPossible$)
input
m: Aktivität, deren Bearbeiter vorgegeben wird
u': der Bearbeiter von Aktivität m
n: Aktivität, für welche die Zulässigkeit des Bearbeiters geprüft werden soll
u: der Bearbeiter von Aktivität n
$PotBearb_n$: die Menge der potentiellen Bearbeiter der Aktivität n
result
boolean: True genau dann, wenn die angegebene Kombination von Bearbeitern zulässig ist
begin
if $u \notin PotBearb_n$ **then return** False;
if $Dep(n,m) = B$ **then**
return $u = u'$;
else if $Dep(n,m) = OE$ **then**
return $OE(u) = OE(u')$;
else // $Dep(n,m) = NULL$
return True;
end.

5.1.5 Automatische Berechnung geeigneter Serverzuordnungen

Im Folgenden wird untersucht, wie geeignete variable Serverzuordnungsausdrücke für die Aktivitäten einer WF-Vorlage systemunterstützt ermittelt werden können. Dabei konzentrieren wir uns im Moment auf die Verteilungsalgorithmen. In Abschnitt 5.3 gehen wir dann auf die Auswirkung von variablen Serverzuordnungen auf das Kostenmodell und auf die Berechnung der WV ein.

Prinzipiell können zum Berechnen von geeigneten Serverzuordnungen weiterhin die in Abschnitt 4.5 vorgestellten Verteilungsalgorithmen verwendet werden, indem deren Eingabeparameter $PotServZuordn_n$ mit geeigneten Werten belegt wird. Dieser Eingabeparameter gibt für jede Aktivität n der WF-Vorlage an, welche Ausdrücke als Serverzuordnung für die Aktivität in Frage kommen. Während $PotServZuordn_n$ im statischen Fall die Menge der WF-Server repräsentiert, muss nun die Menge der für die Aktivität n möglichen variablen Serverzuordnungsausdrücke angegeben werden. Dies sind weiterhin die statischen Serverzuordnungen (Typ 1), sowie Ausdrücke vom Typ 2-5 aus Abschnitt 5.1.2, wobei für alle von Aktivität n referenzierten Aktivitäten m die Bedingung $RefAllowed(CFS, n, m) = True$ gelten muss. Mit diesen Werten für $PotServZuordn_n$ kann der Algorithmus 4.2, welcher alle Kombinationen von Serverzuordnungen „durchprobiert“, unverändert weiterverwendet werden. Dieser ermittelt dann die optimalen Serverzuordnungen für die betrachtete WF-Vorlage. Dieser Algorithmus ist allerdings schon bei rein statischen Serverzuordnungen wegen seiner hohen Laufzeitkomplexität kaum einsetzbar, da er alle Kombinationen von möglichen Serverzuordnungen der Aktivitäten durchspielt. Dieser Schwachpunkt ist bei Berücksichtigung variabler Serverzuordnungen noch dramatischer: In den Serverzuordnungen der $|N|$ Aktivitäten können im Extremfall alle Vorgängeraktivitäten referenziert werden, so dass sich für den Algorithmus eine Komplexität von $O(|N|^{|N|})$ ergibt.

Wegen der hohen Laufzeitkomplexität des Verteilungsalgorithmus 4.2 sollte einer der Algorithmen 4.3 - 4.5 zur Berechnung von variablen Serverzuordnungen eingesetzt werden. Diese Algorithmen haben eine wesentlich bessere Laufzeitkomplexität, verwenden aber den „Trick“ des Zusammenfassens von Partitionen. Sie müssen bei Verwendung variabler Serverzuordnungen leicht modifiziert werden, weil die Veränderung der Serverzuordnung einer Aktivität ungewollte Auswirkungen auf Aktivitäten ha-

ben kann, welche diese Aktivität in ihrer Serverzuordnung referenzieren. Welche Modifikationen der Algorithmen notwendig sind, wird im Folgenden beschrieben.

5.1.5.1 Potentielle Serverzuordnungen einer Aktivität

Die Algorithmen 4.3 - 4.5 gehen in zwei Phasen vor. In der Phase 1 wird jede Aktivität isoliert betrachtet und es wird die jeweils optimale Serverzuordnung berechnet. Prinzipiell können dabei in $ServZuordn_n$ alle Aktivitäten m referenziert werden, für die $RefAllowed(CFS, n, m)$ gilt. Allerdings macht die Referenzierung einer Aktivität m, die sich nicht in $RelevantAct_n$ befindet, keinen Sinn, da keine Beziehung zwischen den Bearbeitern der Aktivitäten m und n besteht. Durch eine solche Referenzierung kann sich keine optimale Serverzuordnung ergeben. Deshalb genügt es, für $ServZuordn_n$ die Referenzierung der relevanten Aktivitäten $RelevantAct_n$ zu betrachten. Damit ergibt sich $PotServZuordn_n$ als die Menge der statischen Serverzuordnungen vereinigt mit Serverzuordnungsausdrücken vom Typ 2-5, welche Aktivitäten aus der Menge $RelevantAct_n$ referenzieren. Da $RelevantAct_n$ i.d.R. eine wesentlich kleinere Menge darstellt als $\{m \mid RefAllowed(CFS, n, m)\}$, müssen deutlich weniger Serverzuordnungen betrachtet werden, was den Aufwand für die Phase 1 der Algorithmen drastisch reduziert. Da nur solche Serverzuordnungen ignoriert werden, die zu keiner optimalen Lösung führen können, wird die Qualität des Ergebnisses durch diese Optimierung nicht beeinträchtigt.

In der Phase 2 der Algorithmen 4.3 - 4.5 werden auch Migrationskosten berücksichtigt. Beim dafür notwendigen Zusammenfassen von Partitionen kann sich für eine Aktivität n auch eine Serverzuordnung ergeben, die bei isolierter Betrachtung der Aktivität n nicht sinnvoll ist. Solen z.B. die Aktivitäten der Partition P mit der Aktivität m zusammengefasst werden, so kann sich für die Aktivitäten $n \in P$ der Serverzuordnungsausdruck $ServZuordn_n =$ `Server(m)` ergeben. Dabei ist es durchaus möglich, dass $ServZuordn_n =$ `Server(m)` für eine Aktivität $n \in P$ isoliert betrachtet nicht sinnvoll ist (weil $m \notin RelevantAct_n$), dass aber durch das Zusammenfassen der Partitionen hohe Migrationskosten eingespart werden. Deshalb müssen in der Phase 2 durch $ServZuordn_n$ auch Aktivitäten $m \notin RelevantAct_n$ referenziert werden dürfen. Es ist aber nicht erlaubt, eine beliebige Aktivität m zu referenzieren. Die Aktivität m muss garantiert vor n ausgeführt werden, um sicherzustellen, dass die Serverzuordnung zur Ausführungszeit ausgewertet werden kann. Beim Zusammenfassen muss also geprüft werden, ob die in $ServZuordn_n$ referenzierte Aktivität überhaupt referenziert werden darf, d.h., ob für $m = ReferencedAct_n$ die Bedingung $RefAllowed(CFS, n, m) = True$ erfüllt ist. Diese Prüfung muss für alle Aktivitäten n der Partition P durchgeführt werden, weil alle diese Aktivitäten $n \in P$ die Aktivität m nach dem Zusammenfassen referenzieren werden. Der vollständige Algorithmus, der sich durch diese (und weitere) Überlegungen ergibt, wird in Abschnitt 5.1.5.4 vorgestellt.

5.1.5.2 Abhängige Aktivitäten

Beim Zusammenfassen von Partitionen werden die Serverzuordnungen der Aktivitäten $n \in P$ verändert. Dies kann unerwünschte Auswirkungen auf eine Aktivität l haben, die später im Ablauf folgt. Ein solches Beispiel ist in Abb. 5.6 dargestellt.

Soll die Aktivität l nicht vom Zusammenfassen der Partition P mit ihrer Vorgängerpartition betroffen sein ($l \notin P$), so darf der tatsächlich für diese Aktivität resultierende Server nicht durch das Zusammenfassen verändert werden. Für die Serverzuordnung der Aktivität l sind folgende Problemfälle

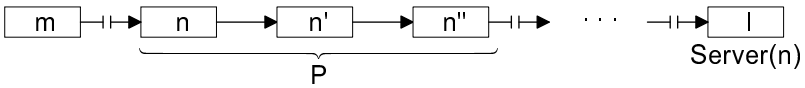


Abbildung 5.6 Wird die Partition P dem Server der Aktivität m zugeordnet, so ändert sich der Server der an dieser Veränderung eigentlich nicht beteiligten Aktivität l.

denkbar, weil sich bei ihnen die Serverzuordnung der Aktivität l auf den (durch das Zusammenfassen veränderten) Server einer Aktivität $n \in P$ bezieht:

1. $ServZuordn_l =$ `Server(n)`
2. $ServZuordn_l =$ `f(Server(n))`

Der Algorithmus 5.4 berechnet für die Aktivität l eine Serverzuordnung $ErgServZuordn$, so dass, trotz veränderter Serverzuordnungen der Aktivitäten $n \in P$, der Server von Aktivität l unverändert bleibt. Im ersten oben beschriebenen Problemfall lässt sich Unabhängigkeit von der Änderung erreichen, indem für die Aktivität l die ursprüngliche Serverzuordnung der referenzierten Aktivität n verwendet wird. Dadurch wird die Aktivität l von demjenigen Server kontrolliert, der ursprünglich für die Aktivität n und damit auch für l geplant war. Deshalb ergibt sich im Algorithmus 5.4 $ErgServZuordn$ als $ServZuordn_n$. Lautet $ServZuordn_l =$ `f(Server(m))`, so kann $ServZuordn_n$ nicht direkt übernommen werden. Wird auf $ServZuordn_n$ aber zusätzlich die Funktion f angewandt, so bleibt der Server von Aktivität l unverändert. Bei allen anderen Serverzuordnungen von Aktivität l kann diese unverändert übernommen werden, da keiner der oben beschriebenen Problemfälle vorliegt.[3] Die gesamte Berechnung der neuen Serverzuordnung für Aktivität l ist in eine Schleife eingebettet (loop). Diese wird erst dann verlassen (break), wenn für die Aktivität l keiner der oben beschriebenen Problemfälle mehr vorliegt. Eine mehrstufige Ersetzung von $ServZuordn_l$ kann notwendig werden (vgl. Abb. 5.6), wenn die Aktivität l von einer Aktivität $n' \in P$ abhängig ist, und n' wiederum von einer Aktivität $n \in P$ abhängt (z.B. $ServZuordn_{n'} =$ `Server(n)`). Wenn der Algorithmus 5.4 verwendet wird, dann entstehen durch das Zusammenfassen von Partitionen also keine unerwünschten Seiteneffekte auf Aktivitäten $l \notin P$. Deshalb gibt es keine Fälle, in denen das Zusammenfassen von Partitionen durch solche Seiteneffekte unmöglich gemacht wird.

Die Funktionsweise von Algorithmus 5.4 soll noch an einem Beispiel verdeutlicht werden: Angenommen die Serverzuordnung der Aktivität n wird durch das Zusammenfassen verändert und es gilt $ServZuordn_l =$ `Server(n)` und $ServZuordn_n =$ `Domain(Bearb(m))`. Dann erzeugt der Algorithmus die Serverzuordnung $ServZuordn_l =$ `Domain(Bearb(m))` für die Aktivität l, welche direkt die Aktivität m referenziert. Bei den gegebenen Serverzuordnungen ist sichergestellt, dass die Aktivität n stets ausgeführt wird, falls die Aktivität l ausgeführt wird, und dass die Aktivität m stets ausgeführt wird, falls die Aktivität n ausgeführt wird. Andernfalls wären die ursprünglichen Serverzuordnungen nicht zulässig gewesen. Damit ist aber auch sichergestellt, dass die Aktivität m stets ausgeführt wird, falls die Aktivität l ausgeführt wird. Deshalb ist eine Überprüfung der Zulässigkeit der resultierenden Serverzuordnung ($RefAllowed(CFS, l, m) = True$) in Algorithmus 5.4 nicht erforderlich.

[3] Wird in einer Serverzuordnung vom Typ 6 der Server einer Aktivität $n \in P$ referenziert (Problemfall 1 oder 2), so muss auch dieser Teil der Serverzuordnung (wie beschrieben) ersetzt werden.

Algorithmus 5.4 (Kopieren einer Serverzuordnung: $CopySZ$)
input
l: Aktivität, deren Serverzuordnung unverändert „kopiert“ werden soll
P: Partition, deren Aktivitäten einem anderen Server zugeordnet werden
$ServZuordn_n$: $\forall$ Aktivitäten n: die ursprüngliche Serverzuordnung der Aktivität n
output
$ErgServZuordn$: eine Serverzuordnung für die Aktivität l, die unabhängig von der Partition P ist
begin
loop
case $ServZuordn_l =$ `Server(n)`: // Problemfall 1
if $n \in P$ **then**
$ErgServZuordn = ServZuordn_n$;
$ServZuordn_l = ErgServZuordn$;
else
$ErgServZuordn = ServZuordn_l$;
break;
case $ServZuordn_l =$ `f(Server(n))`: // Problemfall 2
if $n \in P$ **then**
$ErgServZuordn =$ `f(`$ServZuordn_n$`)`;
$ServZuordn_l = ErgServZuordn$;
else
$ErgServZuordn = ServZuordn_l$;
break;
default case: // Serverzuordnung vom Typ 1, 3, 5 oder 6[3]
$ErgServZuordn = ServZuordn_l$;
break;
end.

5.1.5.3 Zusammenfassen mit nachfolgenden Partitionen

Das Zusammenfassen einer Partition P mit einer nachfolgenden Aktivität m ist nicht immer möglich (vgl. Abb. 5.7). Es kann vorkommen, dass die zur Auswertung von $ServZuordn_m$ notwendigen Laufzeitdaten, zum Zeitpunkt der Ausführung der ersten Aktivität n dieser Partition P, noch nicht existieren. Dieser Fall tritt auf, wenn sich $ServZuordn_m$ auf den Server oder Bearbeiter einer Aktivität l bezieht, die keine Vorgängeraktivität von n ist. Beim Zusammenfassen entlang einer normalen Kontrollkante tritt dieser Fall auf, wenn $l \in P$ gilt (Abb. 5.7a). Sollen Aktivitäten paralleler Zweige entlang einer Synchronisationskante zusammengefasst werden (Abb. 5.7b), so handelt es sich bei l um eine Aktivität eines parallelen Zweiges. Im Falle eines Zusammenfassens entlang einer Schleifenrücksprungkante (Abb. 5.7c) tritt ein Problem auf, wenn l eine Nachfolgeraktivität von P ist. In all diesen Fällen ist diese Art des Zusammenfassens nicht möglich. Im Fall a und c kann die Migration trotzdem stets verhindert werden, indem man der Partition von Aktivität m den Server von P zuordnet, also „andersherum“ zusammenfasst.

Das vorliegende Problem, eine Partition P vom selben Server kontrollieren zu lassen, wie die Aktivität m, kann für jede Aktivität $n \in P$ einzeln gelöst werden: $\forall n \in P$ muss die Aktivität n vom selben Server kontrolliert werden, wie die Aktivität m. Der Algorithmus 5.5 versucht für die Aktivität n eine Serverzuordnung zu berechnen, durch welche n vom selben Server kontrolliert wird, wie die Aktivität m. Eine Aktivität n dem Server einer Vorgängeraktivität m zuzuordnen (welche garantiert ausgeführt wird) ist stets möglich, indem die Serverzuordnung $ServZuordn_n =$ `Server(m)` verwendet wird. Schwieriger ist der Fall, dass m keine solche Vorgängeraktivität von n ist: Wird für die

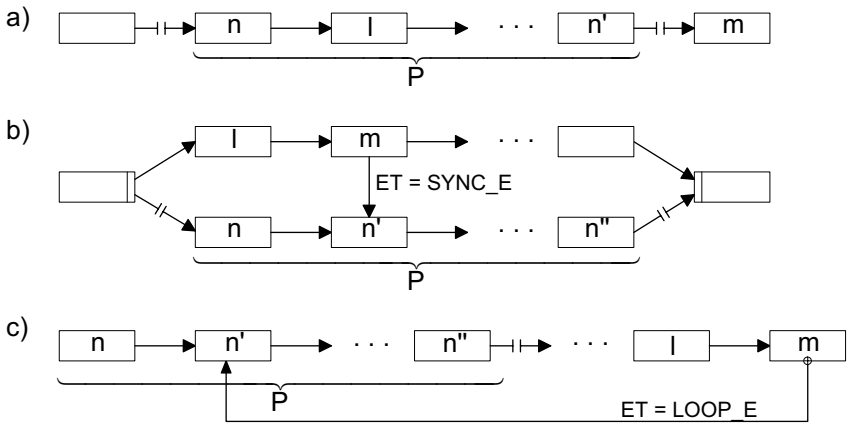


Abbildung 5.7 Die Partition P soll vom Server der Aktivität m kontrolliert werden.

Aktivität m eine statische Serverzuordnung verwendet, so kann diese für n übernommen werden. Lautet $ServZuordn_m =$ `Server(l)`, so werden die Aktivitäten m und l vom selben Server kontrolliert. Obwohl in $ServZuordn_n$ die Aktivität m nicht referenziert werden kann, besteht die Möglichkeit, dass deren Vorgängeraktivität l referenziert werden kann. Deshalb wird durch den rekursiven Aufruf der Funktion $ModifySZ$ überprüft, ob diese Serverzuordnung erlaubt ist. Wird die Aktivität m vom Server im Domain des Bearbeiters einer Aktivität l kontrolliert, so ergibt sich für die Aktivitäten n und m derselbe Server, wenn $ServZuordn_n =$ `Domain(Bearb(l))` verwendet wird. Dies ist aber nur möglich, falls die Aktivität l in $ServZuordn_n$ referenziert werden darf. Andernfalls kann die Aktivität n dem Server der Aktivität m nicht zugeordnet werden, d.h. das geplante Zusammenfassen ist nicht möglich. Enthält $ServZuordn_m$ eine Funktion f, so muss diese auch auf $ErgServZuordn$ angewandt werden; ansonsten verläuft die Berechnung wie bei den vorherigen beiden Fällen. Wird als $ServZuordn_m$ ein vom Modellierer vorgegebener Serverzuordnungsausdruck verwendet, der nicht den Typen 1-5 aus Abschnitt 5.1.2 entspricht, so kann die Semantik dieses Ausdrucks nicht analysiert werden, so dass das gewünschte Zusammenfassen nicht möglich ist. Der Ausdruck kann nicht einfach für Aktivität n übernommen werden, da die Werte von darin verwendeten Datenelementen bei der Ausführung von Aktivität n evtl. noch nicht feststehen.

5.1.5.4 Verteilungsalgorithmus

Wie wir in den vorherigen Unterabschnitten gesehen haben, kann bei Verwendung variabler Serverzuordnungen in Phase 2 der Algorithmen 4.3 - 4.5 eine zu analysierende Verteilung ($TestServZuordn$) nicht durch das direkte Übernehmen von Serverzuordnungen erzeugt werden. Stattdessen müssen die Funktionen $ModifySZ$ und $CopySZ$ verwendet werden, um $TestServZuordn$ zu erzeugen. Damit ergibt sich für Algorithmus 4.3 die in Algorithmus 5.6 dargestellte Phase 2 (analog für die Algorithmen 4.4 und 4.5).

5.2 Verhalten zur Ausführungszeit der Workflow-Instanzen

Nachdem in Abschnitt 5.1 die variablen Serverzuordnungsausdrücke eingeführt wurden, wird im Folgenden auf die Ausführung von WF-Instanzen mit variablen Serverzuordnungen eingegangen. Da

Algorithmus 5.5 (Serverzuordnung verändern: $ModifySZ$)
input
$CFS = (N, \ldots)$: Kontrollflussstruktur des WF-Typs
n: Aktivität, deren Serverzuordnung verändert werden soll
m: der Server der Aktivität m soll auch die Aktivität n kontrollieren
$ServZuordn_n$: $\forall$ Aktivitäten n: die ursprüngliche Serverzuordnung der Aktivität n
output
$ErgServZuordn$: die für die Aktivität n neu berechnete Serverzuordnung
result
boolean: True genau dann, wenn das Zusammenfassen möglich ist
begin
// Aktivität m darf in $ServZuordn_n$ referenziert werden
if $RefAllowed(CFS, n, m)$ **then**
$ErgServZuordn =$ `Server(m);`
return True;
// Aktivität m darf in $ServZuordn_n$ nicht referenziert werden
case $ServZuordn_m =$ `s`:
$ErgServZuordn =$ `s;`
return True;
case $ServZuordn_m =$ `Server(l)`:
return $ModifySZ(CFS, n, l, ServZuordn_*, ErgServZuordn)$;
case $ServZuordn_m =$ `Domain(Bearb(l))`:
if $RefAllowed(CFS, n, l)$ **then**
$ErgServZuordn =$ `Domain(Bearb(l));`
return True;
else
return False;
case $ServZuordn_m =$ `f(Server(l))`:
ZusammenfassenMöglich = $ModifySZ(CFS, n, l, ServZuordn_*, ErgServZuordn')$;
if *ZusammenfassenMöglich* **then**
$ErgServZuordn =$ `f(`$ErgServZuordn'$`);`
return True;
else
return False;
case $ServZuordn_m =$ `f(Domain(Bearb(l)))`:
if $RefAllowed(CFS, n, l)$ **then**
$ErgServZuordn =$ `f(Domain(Bearb(l)));`
return True;
else
return False;
default case: // beliebiger vom Modellierer vorgegebener Ausdruck
return False;
end.

variable Serverzuordnungen wesentlich komplexer sind als statische, sind einige zur Ausführungszeit entstehende Probleme zu lösen, die bei statischen Serverzuordnungen nicht auftreten.

Generell funktioniert die Ausführung von WF-Instanzen, welche variable Serverzuordnungen verwenden, ebenso wie schon für statische Serverzuordnungen in Abschnitt 3.3 beschrieben. Das heißt, vereinfacht gesprochen, wenn eine Nachfolgeraktivität der soeben beendeten Aktivitäteninstanz von einem fremden WF-Server kontrolliert wird, so wird die WF-Instanz zu diesem Server migriert. Der

Algorithmus 5.6 (Erweiterte Phase 2 von Algorithmus 4.3)

...

Phase 2:

$MinCost = calculate\_costs(ServZuordn_*, all)$;

for *PartGröße* = 1 **to** $|N|$ **do**

for each $P : |P| =$*PartGröße*

$\wedge$ P ist max. Teilgraph mit $\forall l_1, l_2 \in P : ServZuordn_{l_1} \equiv ServZuordn_{l_2}$ **do**

$OptAct = NULL$;

for each $a \notin P$ mit $\exists\, l \in P$:

$a \in Pred_{\{CONTROL\_E, SYNC\_E, LOOP\_E\}}(l) \vee$

$a \in Succ_{\{CONTROL\_E, SYNC\_E, LOOP\_E\}}(l)$ **do**

ZusammenfassenMöglich $= True$;

for each $l \in P$ **do**

if not $ModifySZ(CFS, l, a, ServZuordn_*, TestServZuordn_l)$ **then**

ZusammenfassenMöglich $= False$;

if *ZusammenfassenMöglich* **then**

for each $l \notin P$ **do**

$CopySZ(l, P, ServZuordn_*, TestServZuordn_l)$;

$TestCost = calculate\_costs(TestServZuordn_*, all)$;

if $TestCost < MinCost$ **then**

$OptAct = a$;

$MinCost = TestCost$;

if $OptAct \neq NULL$ **then**

for each $l \in P$ **do** $ModifySZ(CFS, l, OptAct, ServZuordn_*, ServZuordn_l)$;

for each $l \notin P$ **do** $CopySZ(l, P, ServZuordn_*, ServZuordn_l)$;

end.

Unterschied zu statischen Serverzuordnungen besteht nun darin, dass dieser Server erst durch die Auswertung eines Serverzuordnungsausdrucks ermittelt werden muss. Da sich Serverzuordnungsausdrücke aber nur auf Vorgängeraktivitäten beziehen dürfen (vgl. Abschnitt 5.1.4.1), kann ein Serverzuordnungsausdruck stets ausgewertet werden, wenn die entsprechende Aktivität aktiviert werden soll, weil dann alle Vorgängeraktivitäten beendet sein müssen.

Durch die Verwendung variabler Serverzuordnungen ergibt sich allerdings an Synchronisationspunkten (Aktivitäten an denen parallele Zweige zusammengeführt werden oder bei denen Synchronisationskanten eingehen) eine Schwierigkeit. Da der Serverzuordnungsausdruck einer Synchronisationsaktivität von einer beliebigen Vorgängeraktivität abhängig sein kann, kann der entsprechende Server evtl. nicht von den Servern aller Vorgängeraktivitäten ermittelt werden. In dem in Abb. 5.8 dargestellten Beispiel kann nach Beendigung der Aktivität c von dem zugehörigen Server nicht bestimmt werden, welcher Server die Nachfolgeraktivität e kontrolliert. Dieser Server hängt nämlich von Daten (dem Bearbeiter der Aktivität d) des unteren Zweiges ab, die im oberen Zweig nicht bekannt sind, da die Aktivität d von einem anderen WF-Server kontrolliert wird. Deshalb kann die Migration von Aktivität c nach e nicht ohne zusätzliche Maßnahmen durchgeführt werden. Im Folgenden wird betrachtet, welche Verfahren zur Lösung dieses Problems geeignet sind.

5.2.1 Mögliche Lösungsansätze

Welche Möglichkeiten gibt es nun, die benötigte Information (Zielserver der Migration) den Quellservern zur Verfügung zu stellen? Die ID des Zielservers direkt an alle betroffenen Quellserver der

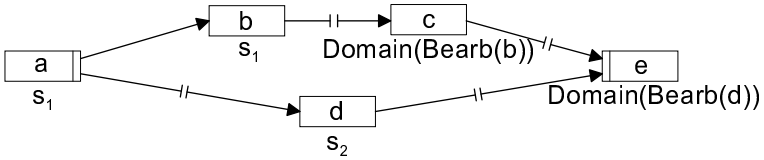


Abbildung 5.8 Beispiel für einen WF, bei dem der Server der Join-Aktivität e nicht in allen parallelen Zweigen berechnet werden kann.

Migration zu senden, ist leider nicht möglich, da diese ihrerseits nicht immer allgemein bekannt sind. So ist in Abb. 5.8 nach Beendigung von Aktivität d im unteren Zweig nicht bekannt, welcher Server die Aktivität c im oberen Zweig kontrolliert, weil auch für die Aktivität c eine variable Serverzuordnung verwendet wird.

Die benötigte Information kann natürlich verbreitet werden, indem beim Erreichen einer Synchronisationsaktivität der entsprechende Server per Broadcast allen anderen Servern des WfMS bekannt gegeben wird. Dies führt in großen Systemen allerdings zu sehr vielen unnötigen Nachrichten. Eine weitere Möglichkeit ist, einen bestimmten (zentralen) Server zur Informationsvermittlung zu verwenden. Dies verschlechtert aber die Verfügbarkeit des WfMS, weil es blockiert ist, wenn diese zentrale Informationsvermittlung ausfällt. Da zu einem fest gewählten Punkt kommuniziert wird, wird außerdem auch dann weit entfernte Kommunikation notwendig, wenn sich alle Server und die Clients einer WF-Instanz in einem räumlich beschränkten Gebiet befinden. Der im nächsten Abschnitt vorgestellte Ansatz vermeidet diese Nachteile.

5.2.2 Workflow-Server einer Synchronisationsaktivität

Um ein günstiges Kommunikationsverhalten zu erhalten, ist es vorteilhaft, die Struktur des WF zum Verbreiten der benötigten Information auszunutzen: Der Algorithmus 5.7 beschreibt den Ablauf, der vor der Aktivierung der Aktivität $NextAct$ vom Server s_{local}, welcher die WF-Instanz aktuell kontrolliert, durchgeführt werden muss. Kann dieser den WF-Server s der Aktivität $NextAct$ berechnen, so wird mit Hilfe der WF-Vorlage die Menge V der direkten Vorgängeraktivitäten von $NextAct$ berechnet. Den Servern dieser Aktivitäten muss der Server s bekannt gemacht werden, sofern sie ihn nicht selbst ermitteln können. Gibt es in der Menge V eine Aktivität v, die keine Nachfolgeraktivität der in $ServZuordn_{NextAct}$ referenzierten Aktivität m ist, so kann der Server s vom Server der Aktivität v nicht berechnet werden. In diesem Fall wird ihm s durch folgendes Verfahren mitgeteilt: s wird an einen Dispatcher gemeldet ($NotifyMigration$), der über einen Zuordnungsausdruck (ähnlich wie die WF-Server) für jede Synchronisationsaktivität festgelegt ist. Bei diesem Dispatcher kann der WF-Server der Aktivität v den Server s erfragen. Welcher Server die Rolle des Dispatchers übernimmt, wird durch den Zuordnungsausdruck $Dispatcher_{NextAct}$ festgelegt. Damit dieser von dem Server auch ausgewertet werden kann, muss der Zuordnungsausdruck statisch sein oder es muss eine Aktivität referenziert werden, die Vorgänger aller $v \in V$ ist.

Kann der Server s der Aktivität $NextAct$ nicht berechnet werden (Else-Fall), so wird dieser beim Dispatcher erfragt ($InfoMigration$). Verfügt der Dispatcher noch nicht über die benötigte Information, so wird vor der Wiederholung der Anfrage eine gewisse Zeit gewartet. Um den beschrieben Algorithmus (robust) zu realisieren, können weitere (persistente) Migrationszustände MS für Kanten eingeführt werden (vgl. Abschnitt 3.3.2.1). Mit diesen wird beschrieben, in welchem Zustand sich eine Migration gerade befindet, d.h., ob der Zielserver schon an den Dispatcher gemeldet wurde, oder ob er schon von diesem erfragt wurde (für Details siehe [Zei99]).

Algorithmus 5.7 (Aktivierung der Aktivität $NextAct$)
input
$CFS = (N, \ldots)$: Kontrollflussstruktur des WF-Typs
$NextAct$: die zu aktivierende Aktivität
s_{local}: WF-Server, der die WF-Instanz gerade kontrolliert
begin
if ($s = Server_{NextAct}$) kann berechnet werden **then**
// Server von $NextAct$ bekannt geben (nur falls von anderen Servern benötigt)
$m = ReferencedAct_{NextAct}$;
$V = Pred_{\{CONTROL\_E, SYNC\_E\}}(NextAct)$;
if $m \neq NULL \wedge \exists v \in V$: $v \notin Succ^*_{\{CONTROL\_E, SYNC\_E\}}(m) \cup \{m\}$ **then**
$NotifyMigration(InstID, NextAct, s) \longrightarrow Dispatcher_{NextAct}$;
else
// WF-Server der Aktivität $NextAct$ beim $Dispatcher_{NextAct}$ erfragen
Nachfragen: $s = InfoMigration(InstID, NextAct) \longrightarrow Dispatcher_{NextAct}$;
if $s = unknown$ **then wait**; **goto** Nachfragen;
Migration durchführen, falls ein Wechsel des WF-Servers stattfindet
if $s \neq s_{local}$ **then**
$Migrate(Instance) \longrightarrow s$;
end.

Das in Algorithmus 5.7 beschriebene Verfahren ist recht effizient, da nur wenige sehr kleine Nachrichten ausgetauscht werden. Es kann aber noch optimiert werden: Um Kommunikationen einzusparen, ist es möglich, Anfragen an den Dispatcher Huckepack [Tan92] mit anderen Nachrichten zu transportieren. Um wiederholte Aufrufe von $InfoMigration$ zu vermeiden, kann die Schleife weggelassen werden, wenn sich der Dispatcher noch nicht beantwortbare Anfragen merkt und sie beantwortet, sobald er die benötigte Information besitzt.

Wird in dem in Abb. 5.9 dargestellten Beispiel der untere Zweig der bedingten Verzweigung (mit Aktivität f) gewählt, so ergibt sich eine Schwierigkeit bei der Signalisierung der Synchronisationskante $b \rightarrow e$. Da die Aktivität e überhaupt nicht ausgeführt wird, gibt es auch keinen WF-Server, der ihre Ausführung kontrolliert. Das Problem ist nun, dass der Server von Aktivität b in der Schleife von Algorithmus 5.7 ständig versucht, den Server von Aktivität e vom $Dispatcher_e$ zu erfragen. Damit dieser Vorgang terminiert, muss auch dann ein Server an den Dispatcher gemeldet werden, wenn die Aktivität e nicht ausgeführt wird, sondern den Zustand SKIPPED erreicht[4]. Für solche Aktivitäten sollte derjenige Server an den Dispatcher gemeldet werden, der den Zustand SKIPPED ermittelt hat, hier also der Server der Split-Aktivität d. Nachdem diese Information an den $Dispatcher_e$ übermittelt wurde, wird die Schleife von Algorithmus 5.7 verlassen, so dass die Synchronisationskante $b \rightarrow e$ signalisiert werden kann. Der Algorithmus 5.7 muss also sowohl für Aktivitäten $NextAct$ aufgerufen werden, deren Zustand in ACTIVATED verändert werden soll, als auch für solche, deren Zustand in SKIPPED verändert wird.

Es bleibt noch zu klären, wie der Zuordnungsausdruck für den Dispatcher der Aktivität $NextAct$ gewählt werden soll. Da mit dem Dispatcher nur kleine Nachrichten ausgetauscht werden, ist es eine akzeptable Möglichkeit, die (durch den Verteilungsalgorithmus ermittelte) beste statische Serverzuordnung der entsprechenden Aktivität zu verwenden. Eine ebenfalls einfach zu realisierende Alternative stellt die Verwendung des Startservers der WF-Instanz dar, da dieser allen an der WF-Instanz

[4]Dies muss zumindest dann erfolgen, wenn die Aktivität mehr als eine eingehende Kante vom Typ $CONTROL\_E$ und $SYNC\_E$ hat.

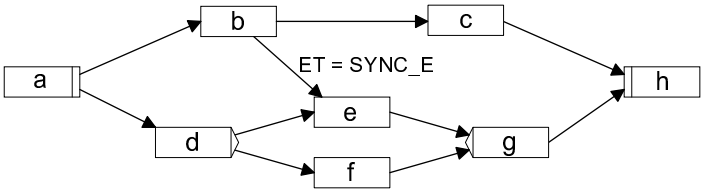


Abbildung 5.9 Synchronisation mit einer Aktivität aus eine bedingten Verzweigung.

beteiligten Servern bekannt ist. Um eine hohe Lokalität zwischen den Aktivitäten der Vorgängermenge V und dem Dispatcher zu erhalten, ist es jedoch günstiger, den Server einer Aktivität zu wählen, welche im WF-Ablauf nahe bei $NextAct$ liegt (in der Hoffnung, dass sich der WF noch in einem nahe gelegenen Domain befindet). Hierfür bietet sich z.B. die letzte gemeinsame Vorgängeraktivität der Aktivitäten $v \in V$ bzgl. des Kontrollflusses an, da ihr Server den Servern aller Aktivitäten $v \in V$ bekannt ist.

5.3 Wahrscheinlichkeitsverteilungen für Aktivitäten

Wie schon erwähnt wurde, können auch bei variablen Serverzuordnungen die in Abschnitt 4.5 vorgestellten Verteilungsalgorithmen (weitgehend unverändert) verwendet werden. Das von diesen Algorithmen verwendete und in Abschnitt 4.3 aufgestellte Kostenmodell kann sogar völlig unverändert übernommen werden. Von diesem werden allerdings WV benötigt, deren Berechnung für statische Serverzuordnungen in Abschnitt 4.4 erläutert wurde. Bei variablen Serverzuordnungen ist dies, wegen den zwischen den Aktivitäten existierenden Abhängigkeiten, aber wesentlich komplizierter. Im nun folgenden Abschnitt wird beschrieben, wie die WV für die Aktivitätenausführung $ExProb_n(s, D)$ und $\#User_n(D|s)$ berechnet werden können. Die Berechnung der WV für Migrationen $MigrProb_{m,n}(s_1, s_2)$ wird in Abschnitt 5.4 erläutert.

Da variable Serverzuordnungen von Laufzeitdaten der Instanz abhängen, kann dieselbe Aktivität n bei verschiedenen WF-Instanzen von unterschiedlichen Servern kontrolliert werden. Die entsprechende Wahrscheinlichkeit, mit welcher die Aktivität n vom Server s kontrolliert wird, bezeichnen wir mit $\boldsymbol{ServProb_n(s)}$ = *P(Aktivität n wird vom Server s kontrolliert).* Da sich verschiedene WF-Instanzen in unterschiedlichen OE befinden können (und dabei von unterschiedlichen Servern kontrolliert werden können), kann die nachfolgend definierte Bearbeiter-WV für verschiedene Server unterschiedlich sein. Der Anteil an den Bearbeitern von Aktivität n aus dem Domain D, für den Fall, dass der Server s die Aktivität n kontrolliert, wird mit $\boldsymbol{ActorProb_n(D|s)}$ = *P(der Bearbeiter von Aktivität n befindet sich im Domain D | der Server der Aktivität n befindet sich im Domain s)* bezeichnet. Das Hauptproblem bei der Berechnung der vom Kostenmodell benötigten WV $ExProb_n(s, D)$ besteht nun darin, für gegebene Serverzuordnungen die WV $ServProb_n(s)$ und $ActorProb_n(D|s)$ zu berechnen. Sind diese erst bekannt, so kann $ExProb_n(s, D)$ leicht berechnet werden: $ActorProb_n(D|s)$ muss dazu nur noch mit der Wahrscheinlichkeit $ServProb_n(s)$ gewichtet werden, mit der Server s die Aktivität n kontrolliert. Die Wahrscheinlichkeit, dass der Server s die Aktivität n kontrolliert und ein Bearbeiter im Domain D die Aktivität n bearbeitet, beträgt damit $ExProb_n(s, D) = ServProb_n(s) \cdot ActorProb_n(D|s)$.

Um die Kosten für Arbeitslisten-Updates abschätzen zu können, wird die WV $\#User_n(D|s)$ benötigt – die Anzahl der potentiellen Bearbeiter von Aktivität n im Domain D, falls die Aktivität vom Server s kontrolliert wird. Um diese berechnen zu können, wird die WV $\boldsymbol{\#User_n|s}$ eingeführt: die Anzahl

der Bearbeiter, die Aktivität n bearbeiten dürfen, falls sie vom Server s kontrolliert wird. Da sich diese Bearbeiter entsprechend $ActorProb_n(D|s)$ auf die verschiedenen Domains verteilen, ergibt sich $\#User_n(D|s) = \#User_n|s \cdot ActorProb_n(D|s)$

Für einzelne Aktivitäten können $ServProb_n(s)$ und/oder $ActorProb_n(D|s)$ für bestimmte Serverzuordnungen auch vom Modellierer vorgegeben werden, falls dieser über zusätzliches Wissen verfügt. Ein Beispiel hierfür ist eine Aktivität, die bevorzugt von einem bestimmten Bearbeiter erledigt wird und bei der die anderen potentiellen Bearbeiter nur aushelfen, wenn dieser überlastet ist.

5.3.1 Berechnung der Server-Wahrscheinlichkeitsverteilung $ServProb_n(s)$

Die Server-WV $ServProb_n(s)$ gibt an, mit welcher Wahrscheinlichkeit der Server s die Aktivität n kontrolliert. Sie ergibt sich aus der Serverzuordnung $ServZuordn_n$ und der Server- bzw. Bearbeiter-WV der referenzierten Aktivität m. Diese WV sind bei der für Aktivität n durchgeführten Analyse schon bekannt, weil die Aktivität m im Ablauf vor n liegen muss (sonst darf sie in $ServZuordn_n$ nicht referenziert werden) und die Prozessvorlage von den Verteilungsalgorithmen in partieller Ordnung durchlaufen wird.

Die Berechnung der Server-WV erfolgt durch Algorithmus 5.8 für die in Abschnitt 5.1.2 definierten Serverzuordnungen. Der Fall 6 wird dabei (und im weiteren) nicht berücksichtigt, da bei dieser Serverzuordnung die WV vom Modellierer vorgegeben werden muss. Bei statischer Serverzuordnung wird die Aktivität n stets vom Server s kontrolliert. Deshalb ist $ServProb_n(s') = 1$ für $s = s'$. Ansonsten gilt $ServProb_n(s') = 0$.[5] Wird für die Aktivität n derselbe Server verwendet wie für die Aktivität m, so sind auch die Server-WV gleich. Dieser Aussage liegt die Annahme zugrunde, dass es zwischen m und n keine Verzweigungen gibt, die abhängig von der Wahl der Servers sind. Da solche Zusammenhänge kaum zu fassen sind, da sie Datenelemente betreffen, haben wir für Verzweigungs- und Schleifenbedingungen generell angenommen, dass ihr Ergebnis nicht vom aktuellen Server abhängt, außer der Modellierer gibt dies explizit vor. Lautet die Serverzuordnung `Domain(Bearb(m))`, so wird der Server der Aktivität n im Domain des Bearbeiters der Aktivität m gewählt. Deshalb ergibt sich die Server-WV von Aktivität n aus der Bearbeiter-WV von Aktivität m. Da dabei nicht relevant ist, welcher Server die Aktivität m kontrolliert hat, wird eine vom Server unabhängige Bearbeiter-WV $ActorProb_m(i)$ erzeugt, indem die Bearbeiter-WV der einzelnen Server gewichtet aufaddiert werden. Diese werden dabei jeweils mit der Wahrscheinlichkeit gewichtet, dass der betrachtete Server die Aktivität m kontrolliert. Wird auf die Serverzuordnung zusätzlich noch eine Funktion f angewandt, so muss diese auch auf das Ergebnis angewandt werden. Wie dies exakt funktioniert, zeigt der Algorithmus 5.12 aus Abschnitt 5.3.3.1.

5.3.2 Die Bearbeiter-Wahrscheinlichkeitsverteilung $ActorProb_n(D|s)$

Im Folgenden wird beschrieben, wie die Bearbeiter-WV $ActorProb_n(D|s)$ berechnet werden kann. Bevor wir auf das konkrete Berechnungsverfahren eingehen, werden einige Erweiterungen an der Bearbeitergewichtung diskutiert, die aufgrund der variablen Serverzuordnungen notwendig werden.

[5] $\delta_{i,j}$ ist das Kroneckersymbol [BS89] mit $\delta_{i,j} = 1$, falls $i = j$ und $\delta_{i,j} = 0$, falls $i \neq j$.

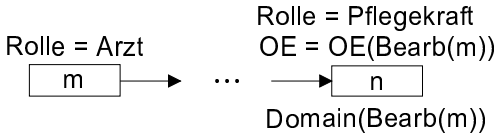
Algorithmus 5.8 (Berechnung der Server-WV $ServProb_n(s)$)
input
n: die Aktivität, für welche die Server-WV berechnet werden soll
$ServZuordn_n$: die für Aktivität n vorgesehene Serverzuordnung
$ServProb_m(s)$: $\forall$ Aktivitäten m: die schon berechneten Server-WV
$ActorProb_m(D|s)$: $\forall$ Aktivitäten m: die schon berechneten Bearbeiter-WV
output
$ServProb_n(s)$: Server-WV für die Aktivität n
begin
case $ServZuordn_n =$ `s`:
$\forall s'$: $ServProb_n(s') = \delta_{s,s'}$;
case $ServZuordn_n =$ `Server(m)`:
$\forall s$: $ServProb_n(s) = ServProb_m(s)$;
case $ServZuordn_n =$ `Domain(Bearb(m))`:
$\forall i$: $ServProb_n(i) = ActorProb_m(i)$
mit $ActorProb_m(D) = \sum_s ActorProb_m(D|s) \cdot ServProb_m(s)$;
case $ServZuordn_n =$ `f(Server(m))`:
$\forall s$: $ServProb_n(s) = f(ServProb_m(s))$;
case $ServZuordn_n =$ `f(Domain(Bearb(m)))`:
$\forall i$: $ServProb_n(i) = f(ActorProb_m(i))$
mit $ActorProb_m(D) = \sum_s ActorProb_m(D|s) \cdot ServProb_m(s)$;
end.

5.3.2.1 Abhängige Gewichte

In Abschnitt 4.2.2.2 wurde für die Bearbeiter ein Gewicht $G_n(u)$ eingeführt, das – vereinfacht ausgedrückt – auf die Aktivität n bezogen angibt, welchen Anteil seiner Arbeitszeit der Benutzer u mit dem WfMS arbeitet. Diese Art der Bearbeitergewichtung muss im Kontext von variablen Serverzuordnungen erweitert werden. In diesem Abschnitt wird motiviert, warum dies notwendig ist, und außerdem wird eine geeignete Erweiterung der Gewichtung vorgestellt.

Außer den benutzerspezifischen Gewichten $G_n(u)$ werden durch die variable Serverzuordnung OE-spezifische Gewichte $G_n(oe)$ notwendig. Der Grund dafür ist, dass nicht mehr angenommen werden kann, dass sich die Bearbeitung der Aktivität n entsprechend der Gewichte $G_n(u)$ auf die Benutzer verteilt. Um dies zu erläutern, nehmen wir an, dass für den in Abb. 5.10 dargestellten Ablauf die angegebene Verteilung von Bearbeitern auf OE gelte. $AnteilBearb_n(oe) = \sum_{u \in PotBearb_n \wedge OE(u)=oe} G_n(u) \;/\; \sum_{u \in PotBearb_n} G_n(u)$ beschreibt dabei den Anteil der Bearbeiter von Aktivität n, die der Organisationseinheit oe angehören.

Obwohl 40% der Bearbeiter von Aktivität n (Pflegekräfte) zur Abteilung 1 gehören, kann nicht angenommen werden, dass 40% dieser Aktivitäten von Bearbeitern der Abteilung 1 ausgeführt werden. Die OE wird nämlich schon bei der Ausführung von Aktivität m festgelegt. Da die Ärzte von Abteilung 1, welche die Aktivität m ausführen, aber nur einen Anteil von 20% haben, wird die Aktivität n ebenfalls nur mit einer Wahrscheinlichkeit von 20% in Abteilung 1 ausgeführt. Der Algorithmus 5.9 berechnet für jede Aktivität n, wie groß der Anteil der einzelnen OE an der Ausführung dieser Aktivität ist ($AnteilAusf_n(oe)$). Ist die Bearbeiterzuordnung der Aktivität n abhängig von einer Vorgängeraktivität m ($Dep(n,m) \in \{B, OE\}$), so bestimmt die Aktivität m den Anteil der einzelnen OE an den Ausführungen der Aktivität n. Andernfalls ($Dep(n,m) = NULL$) entspricht der Anteil einer OE bei der Ausführung ihrem Anteil an den Bearbeitern. Aus den Werten für $AnteilBearb_n(oe)$ und



OE	$\mathbf{AnteilBearb_m}$ (Ärzte)	$\mathbf{AnteilBearb_n}$ (Pflegekräfte)
Abteilung 1	0,2 (2 Bearbeiter mit $G_m(u) = 1$)	0,4 (16 Bearbeiter mit $G_n(u) = 1$)
Abteilung 2	0,5 (5 Bearbeiter mit $G_m(u) = 1$)	0,35 (14 Bearbeiter mit $G_n(u) = 1$)
Abteilung 3	0,3 (3 Bearbeiter mit $G_m(u) = 1$)	0,25 (10 Bearbeiter mit $G_n(u) = 1$)

Abbildung 5.10 Motivation für die Notwendigkeit von OE-spezifischen Gewichten.

$AnteilAusf_n(oe)$ berechnet Algorithmus 5.9 das *OE-Gewicht* $G_n(oe)$, mit dem jeder Bearbeiter dieser OE zusätzlich gewichtet wird. Durch die zusätzliche Gewichtung der Bearbeiter mit diesem OE-Gewicht entspricht der Anteil der Gewichte der Bearbeiter einer OE ihrem Anteil an der Ausführung der Aktivität. Diese Tatsache, die im Anhang B.1 bewiesen wird, zeigt, dass die Berechnung von $Gewicht_n(u)$ auf korrekte Art und Weise erfolgt.

Algorithmus 5.9 (Berechnung der OE-abhängigen Gewichte)
input
$CFS = (N, \ldots)$: Kontrollflussstruktur des WF-Typs
$G_n(u)$: $\forall n \in N$, $\forall$ Benutzer u: das Gewicht des Benutzers u
$PotBearb_n$: $\forall n \in N$: die Menge der potentiellen Bearbeiter der Aktivität n
$Dep(n, m)$: $\forall n, m \in N$: die Beziehung zwischen den Bearbeitern der beiden Aktivitäten
output
$Gewicht_n(u)$: $\forall n \in N$, $\forall$ Benutzer u: das Gesamtgewicht des Benutzers u für Aktivität n
begin
for each Aktivität $n \in N$ (in partieller Ordnung bezüglich Kontrollfluss) **do**
// Anteil der einzelnen OE an den Bearbeitern von Aktivität n berechnen
for each OE oe **do**
$AnteilBearb_n(oe) = \sum_{u \in PotBearb_n \wedge OE(u)=oe} G_n(u) \;/\; \sum_{u \in PotBearb_n} G_n(u)$;
// Anteil der OE an der Ausführung von Aktivität n berechnen
for each OE oe **do**
if $\exists m \in Pred^*_{\{CONTROL\_E, SYNC\_E\}}(n)$ mit $Dep(n, m) \in \{B, OE\}$ **then**
$AnteilAusf_n(oe) = AnteilAusf_m(oe)$
else // $BearbZuordn_n$ ist unabhängige Bearbeiterzuordnung
$AnteilAusf_n(oe) = AnteilBearb_n(oe)$;
$G_n(oe) = AnteilAusf_n(oe)$ / $AnteilBearb_n(oe)$;
// Berechnung der Gesamtgewichte der Bearbeiter
for each $u \in PotBearb_n$ **do**
$oe = OE(u)$;
$Gewicht_n(u) = G_n(oe) \cdot G_n(u)$;
end.

Im Beispiel aus Abb. 5.10 ist für die Abteilung 1 $AnteilAusf_n(Abteilung\ 1) = AnteilAusf_m(Abteilung\ 1) = AnteilBearb_m(Abteilung\ 1) = 0,2$. Damit ergibt sich das OE-Gewicht als $G_n(Abteilung\ 1) = 0,2/0,4 = 0,5$. Für die Pflegekräfte der Abteilung 1 (alle mit $G_n(u) = 1$) ergibt sich damit jeweils das Gesamtgewicht $Gewicht_n(u) = 0,5$.

5.3.2.2 Berechnung der Bearbeiter-WV $ActorProb_n(D|s)$

Die Bearbeiter-WV $ActorProb_n(D|s)$ gibt an, mit welcher Wahrscheinlichkeit der Bearbeiter von Aktivität n aus dem Domain D stammt, wenn die Aktivität n vom Server s kontrolliert wird. Sie ist vom konkret verwendeten Server s dieser Aktivität abhängig. Um dies zu verdeutlichen, stelle man sich ein Krankenhaus mit 3 Abteilungen vor, die jeweils über einen eigenen Server verfügen. Für Patienten der Abteilung 1 sind ausschließlich Pflegekräfte von Abteilung 1 (Domain 1) zuständig. Sinnvollerweise wird in diesem Fall auch der Server 1 verwendet. Damit ergibt sich eine Bearbeiter-WV $ActorProb_n(D|1) = (1, 0, 0)$. Analog ergibt sich für Server 2 die WV $ActorProb_n(D|2) = (0, 1, 0)$ und $ActorProb_n(D|3) = (0, 0, 1)$ für Server 3.

Die Bearbeiter-WV $ActorProb_n(D|s)$ kann durch Algorithmus 5.10 bestimmt werden. Der Algorithmus durchläuft alle Bearbeiter der Aktivität n und ermittelt jeweils den zugehörigen Domain D (aus dem Organisationsmodell). Außerdem bestimmt er den Server s, der die Aktivität kontrolliert, falls dieser Bearbeiter sie ausführt. Dann wird der Bearbeiter u für den entsprechenden Server s und Domain D in der Bearbeiter-WV $ActorProb_n(D|s)$ berücksichtigt. Die Berechnung von $ActorProb_n(D|s)$ erfolgt also entsprechend der Definition der Bearbeiter-WV (siehe Abschnitt 5.3). Ebenso wird der Benutzer u mit seinem entsprechenden Gewicht in der Anzahl der Bearbeiter $\#User_n|s$ berücksichtigt, welche die Aktivität n bearbeiten dürfen, falls diese vom Server s kontrolliert wird. Die Aktivität n kann trotz desselben Bearbeiters durch unterschiedliche Server – jeweils mit einer gewissen Wahrscheinlichkeit – kontrolliert werden. Diese Wahrscheinlichkeiten werden mit der Funktion $DepServ(n, "Bearb = u", \ldots)$ berechnet (siehe Abschnitt 5.3.2.3) und im Vektor $DepServProb_n(s|u)$ gespeichert. Der Bearbeiter wird anteilig bei jedem dieser Server berücksichtigt.

Algorithmus 5.10 (Berechnung der Bearbeiter-WV $ActorProb_n(D|s)$)
input
n: Aktivität, für welche die Bearbeiter-WV berechnet werden soll
$PotBearb_n$: die Menge der potentiellen Bearbeiter der Aktivität n
$Gewicht_n(u)$: $\forall$ Benutzer u: das Gewicht des Benutzers u für die Aktivität n
output
$ActorProb_n(D|s)$: die Bearbeiter-WV für Aktivität n
$\#User_n|s$: Anzahl der pot. Bearbeiter von Aktivität n, falls diese vom Server s kontrolliert wird
begin
$\forall D, s : ActorProb_n(D|s) = 0; \forall s : \#User_n|s = 0;$
for each $u \in PotBearb_n$ **do**
$D = Domain(u);$
$DepServProb_n(s|u) = DepServ(n, "Bearb = u", \ldots);$
for each s **do**
if $DepServProb_n(s|u) \neq 0$ **then**
$ActorProb_n(D|s) = ActorProb_n(D|s) + DepServProb_n(s|u) \cdot Gewicht_n(u);$
$\#User_n|s = \#User_n|s + Gewicht_n(u);$
normalisiere jede Zeile von $ActorProb_n(D|s)$ so, dass $\forall s : \sum_D ActorProb_n(D|s) = 1;$
end.

5.3.2.3 Berechnung der Wahrscheinlichkeitsverteilung $DepServProb_n(s|u)$

Wie im vorherigen Abschnitt erwähnt wurde, muss zur Berechnung der Bearbeiter-WV $ActorProb_n(D|s)$ die vom Bearbeiter abhängige Server-WV $DepServProb_n(s|u)$ berechnet wer-

den, bei welcher der Bearbeiter u der Aktivität n vorgegeben ist. $DepServProb_n(s|u)$ gibt also für einen bestimmten Bearbeiter u an, mit welcher Wahrscheinlichkeit die Aktivität n vom Server s kontrolliert wird. Die Berechnung dieser WV erfolgt auf ähnliche Weise, wie für die bearbeiterunabhängige WV $ServProb_n(s)$ in Abschnitt 5.3.1 beschrieben. Allerdings hängt $DepServProb_n(s|u)$ nicht nur von der Serverzuordnung der betrachteten Aktivität n ab, sondern auch von deren Bearbeiterzuordnung. Algorithmus 5.11 behandelt die bei der Berechnung von $DepServ(n, Cond, \ldots)$ auftretenden Fälle. $Cond$ beschreibt dabei die für Aktivität n betrachtete Bearbeitermenge. Beim Aufruf von Algorithmus 5.11 für den Benutzer u gilt also $Cond =$ "$Bearb = u$".

Zur Erläuterung von Algorithmus 5.11 betrachten wir zuerst die einfachen Fälle: Im Fall 1 ist die Serverzuordnung statisch ($ServZuordn_n =$ `s`), so dass sich stets derselbe Server ergibt. Die Server-WV resultiert damit als $DepServProb_n(s|u) = \delta_{s,s'}$. Ist die Bearbeiterzuordnung unabhängig von anderen Aktivitäten (Fall 2a, 3a, ...),[6] so ist der Server der Aktivität n zwar von der Aktivität m abhängig, nicht aber ihr Bearbeiter. Darum ist der für Aktivität n resultierende Server unabhängig vom vorgegebenen Bearbeiter u der Aktivität n, so dass auch die WV $DepServProb$ unabhängig von diesem Bearbeiter ist. Als Ergebnis ergibt sich deshalb die vom Bearbeiter unabhängige Server-WV $ServProb_n(s)$.

Es bleiben noch die Fälle zu erläutern, bei denen in $ServZuordn_n$ die Aktivität m referenziert wird und zusätzlich der Bearbeiter der Aktivität n von dem Bearbeiter der Aktivität m abhängig ist ($Dep(n, m) \in \{B, OE\}$). Lautet $ServZuordn_n =$ `Server(m)`, so ergibt sich $DepServProb_n(s|u)$ durch Rekursion. Wir betrachten zuerst den Fall, dass die Aktivität n vom selben Benutzer bearbeitet wird wie die Aktivität m (Fall 2b): Da die Aktivitäten n und m in diesem Fall denselben Bearbeiter und denselben Server haben, hat jeder Bearbeiter bei den beiden Aktivitäten auch dieselbe Server-WV. Das Ergebnis ergibt sich also durch direkte Rekursion, wobei als Bedingung $Cond'$ – welche die Bearbeitermenge der Aktivität m beschreibt – die Bedingung $Cond$ unverändert übernommen wird. Wird die Aktivität m lediglich von irgendeinem Benutzer derselben OE bearbeitet wie Aktivität n (Fall 2c), so wird das Ergebnis ebenfalls durch Rekursion berechnet. Dann ist aber nicht garantiert, dass die Aktivitäten n und m vom exakt selben Bearbeiter ausgeführt werden (nur dieselbe OE ist sicher). Deshalb wird die Rekursion mit einer geänderten Bedingung $Cond'$ aufgerufen. Falls $Cond$ einen einzelnen Bearbeiter spezifiziert hat, darf $Cond'$ für die Aktivität m nur die OE des Bearbeiters festlegen. Falls durch $Cond$ lediglich die OE der Bearbeiter vorgegeben ist, kann die Bedingung unverändert für die Rekursion übernommen werden.

Im Fall 3 lautet $ServZuordn_n =$ `Domain(Bearb(m))`. Dann findet keine Rekursion statt, ggf. ist damit das Rekursionsende erreicht. Der Algorithmus durchläuft die Menge der potentiellen Bearbeiter der Aktivität m, da der Server von Aktivität n durch den Domain des Bearbeiters der Aktivität m bestimmt wird, und ermittelt so den Server der Aktivität n. Im Fall 3b haben die Aktivitäten n und m denselben Bearbeiter ($Dep(n, m) = B$), so dass der Bearbeiter der Aktivität m ebenfalls die Bedingung $Cond$ erfüllen muss.[7] Der Algorithmus durchläuft nun alle potentiellen Bearbeiter der Aktivität

[6] Diese Fälle machen scheinbar keinen Sinn, da eine variable Serverzuordnung verwendet wird, obwohl der Bearbeiter der Aktivität n nicht von anderen Aktivitäten abhängt. Sie können aber sehr wohl auftreten, nämlich dann, wenn Aktivitäten zusammengefasst werden um Migrationen einzusparen (Phase 2 der Verteilungsalgorithmen), so dass die Aktivität n in eine Partition mit variablen Serverzuordnungen aufgenommen wird.

[7] Es ist auch möglich, dass durch $Cond$ mehrere Bearbeiter spezifiziert werden. Dies ist z.B. dann der Fall, wenn $DepServ$ (rekursiv) aus dem Fall 2c heraus aufgerufen wurde. Dann spezifiziert $Cond$ lediglich die OE, aus welcher die Bearbeiter der Aktivität n stammen. Wegen $Dep(n, m) = B$ kommen alle diese Benutzer auch für die Bearbeitung der Aktivität m in Frage, so dass sie hier betrachtet werden müssen.

m, die zusätzlich die Bedingung $Cond'$ ($= Cond$) erfüllen. Für jeden dieser Bearbeiter wird der zugehörige Domain ermittelt, da dieser den Server der Aktivität n determiniert. Dieser Domain wird, mit dem Gewicht des entsprechenden Bearbeiters gewichtet, in $SV(s)$ berücksichtigt. Schließlich wird das Ergebnis $SV(s)$ noch normalisiert, so dass die Summe aller Werte 1 ergibt. Diese Vorgehensweise soll nun am Beispiel aus Abb. 5.11a (entspricht dem Fall 3b) noch näher erläutert werden. In diesem Beispiel soll die WV für den Benutzer u = Dr. Brinkmann aus Domain 3 berechnet werden ($Cond =$ "$Bearb = Brinkmann$"). Der Algorithmus durchläuft alle Benutzer u der Aktivität m, welche die Bedingung $Cond$ erfüllen. Er betrachtet also nur Dr. Brinkmann, was korrekt ist, da dieser bei der gegebenen Bearbeiterzuordnung der Aktivität n der einzig mögliche Bearbeiter der Aktivität m ist (weil Dr. Brinkmann als der Bearbeiter der Aktivität n angenommen wird). Für diesen Bearbeiter wird der zugehörige Domain ermittelt, weil dieser den Server der Aktivität n determiniert. Dieser Domain wird (mit dem Gewicht des entsprechenden Bearbeiters) im Ergebnis $SV(s)$ berücksichtigt. Damit ergibt sich für das Beispiel $SV(s) = (0, 0, 1)$, da Dr. Brinkmann zum Domain 3 gehört.

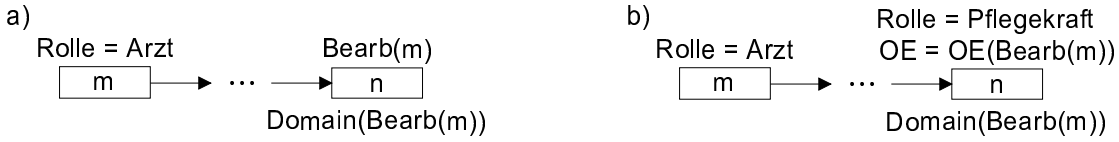


Abbildung 5.11 Beispiele zur Berechnung der abhängigen Server-WV $DepServProb_n(s|u)$.

Im Fall 3c gehören die Bearbeiter der Aktivitäten m und n lediglich derselben OE an. Dann darf die Betrachtung der Bearbeiter nicht auf einen in $Cond$ spezifizierten Bearbeiter eingeschränkt werden. Deshalb wird (wie im Fall 2c) eine Bedingung $Cond'$ erzeugt, welche nur die entsprechende OE spezifiziert. Der weitere Ablauf ist analog zum Fall 3b und wird wieder an einem Beispiel erläutert: In Abb. 5.11b gehören die Bearbeiter der Aktivitäten m und n derselben OE an ($Dep(m, n) = OE$). Soll $DepServProb_n(s|u)$ für Schwester Hildegard aus der OE Abteilung 2 berechnet werden, so werden alle Bearbeiter u (derselben OE Abteilung 2, wegen $Cond' =$ "$oe = Abteilung\ 2$") durchlaufen, welche sich potentiell für die Aktivität m qualifizieren (die Ärzte). Dies sind genau die Bearbeiter, welche die Aktivität m ausgeführt haben können, wenn Aktivität n von Schwester Hildegard bearbeitet wird. Der Domain jedes dieser Ärzte wird im Ergebnis $SV(s)$ mit dem Gewicht des entsprechenden Arztes berücksichtigt, da dieser Domain den WF-Server der Aktivität n determiniert. Schließlich wird das Ergebnis $SV(s)$ wieder normalisiert.

Wird eine Funktion f auf die Serverzuordnung angewandt (Fall 4 und 5), so wird die Server-WV wie in Fall 2 bzw. 3 berechnet und dann auf das Ergebnis die Funktion f angewandt. Wie eine Funktion auf einen Vektor angewandt werden kann, ist in Algorithmus 5.12 (siehe Abschnitt 5.3.3.1) beschrieben.

Hat der Modellierer für die Aktivität n eine Serverzuordnung vorgegeben, die nicht analysiert werden kann (Fall 6 aus Abschnitt 5.1.2), so muss er auch $ServProb_n(s)$ und $ActorProb_n(D|s)$ vorgeben. Da $ActorProb_n(D|s)$ damit schon bekannt ist, wird der Algorithmus 5.10 aus dem vorherigen Abschnitt und damit der Algorithmus 5.11 nicht verwendet. Allerdings kann es vorkommen, dass die benutzerabhängige Serververteilung $DepServProb_n(s|u)$ einer Aktivität mit einer derartigen Serverzuordnung bei einer Rekursion von Algorithmus 5.11 benötigt wird (z.B. für Fall 2b oder 2c). In diesem Fall kann als grobe Näherung die vom Bearbeiter unabhängige Server-WV $ServProb_n(s)$ verwendet werden, die für die Aktivität n vom Modellierer vorgegeben wurde. Hat der Modellierer zusätzlich OE-spezifische Server-WV angegeben, so wird aus $Cond$ die betroffene OE ermittelt, womit die entsprechende Server-WV verwendet werden kann.

Die beschriebenen Mechanismen lassen sich auf weitere Typen von Bearbeiterzuordnungen (siehe

Algorithmus 5.11 (Berechnung von $DepServ(n, Cond, \ldots)$)
input
n: Aktivität, für welche die WV $DepServProb_n(s|u)$ berechnet werden soll
$Cond$: Bedingung, welche die betrachteten Bearbeiter der Aktivität n erfüllen müssen
$ServZuordn_n$: $\forall$ Aktivitäten n: die vorgesehene Serverzuordnung
$PotBearb_n$: $\forall$ Aktivitäten n: die potentiellen Bearbeiter der Aktivität n
$Gewicht_n(u)$: $\forall$ Aktivitäten n, $\forall$ Benutzer u: das Gewicht des Benutzers u für die Aktivität n
$Dep(n, m)$: $\forall$ Aktivitäten n, m: die Beziehung zwischen den Bearbeitern der beiden Aktivitäten
result
$DepServProb_n(s|u)$: die vom spezifizierten Bearbeiter abhängige Server-WV
begin
case $ServZuordn_n =$ `s`: (1)
$\forall s'$: $SV(s') = \delta_{s,s'}$;
case $ServZuordn_n =$ `Server(m)`: (2)
case $Dep(n, m) = NULL$: (2a)
$\forall s$: $SV(s) = ServProb_n(s)$;
case $Dep(n, m) = B$: (2b)
$Cond' = Cond$;
$SV(s) = DepServ(m, Cond', \ldots)$;
case $Dep(n, m) = OE$: (2c)
if $Cond =$ "$Bearb = u$" **then** $oe = OE(u)$; $Cond' =$ "$OE = oe$";
if $Cond =$ "$OE = oe$" **then** $Cond' = Cond$;
$SV(s) = DepServ(m, Cond', \ldots)$;
case $ServZuordn_n =$ `Domain(Bearb(m))`: (3)
case $Dep(n, m) = NULL$: (3a)
$\forall s$: $SV(s) = ServProb_n(s)$;
case $Dep(n, m) = B$: (3b)
$\forall s$: $SV(s) = 0$;
$Cond' = Cond$;
for each $u \in PotBearb_m$ mit u erfüllt $Cond'$ **do**
$s = Domain(u)$;
$SV(s) = SV(s) + Gewicht_m(u)$;
normalisiere SV(s) so, dass $\sum_s SV(s) = 1$;
case $Dep(n, m) = OE$: (3c)
$\forall s$: $SV(s) = 0$;
if $Cond =$ "$Bearb = u$" **then** $oe = OE(u)$; $Cond' =$ "$OE = oe$";
if $Cond =$ "$OE = oe$" **then** $Cond' = Cond$;
for each $u \in PotBearb_m$ mit u erfüllt $Cond'$ **do**
$s = Domain(u)$;
$SV(s) = SV(s) + Gewicht_m(u)$;
normalisiere SV(s) so, dass $\sum_s SV(s) = 1$;
case $ServZuordn_n =$ `f(Server(m))`: (4)
$\ldots$ // wie Fall 2
if $(Dep(n, m) \neq NULL$ **then** $SV(s) = f(SV(s))$; // siehe Algorithmus 5.12
case $ServZuordn_n =$ `f(Domain(Bearb(m)))`: (5)
$\ldots$ // wie Fall 3
if $(Dep(n, m) \neq NULL$ **then** $SV(s) = f(SV(s))$; // siehe Algorithmus 5.12
return SV(s);
end.

Abschnitt 2.4.3.1) erweitern. Für diese können weitere Arten von Abhängigkeiten und weitere Bearbeiterbedingungen $Cond$ definiert werden. Soll z.B. die Aktivität n vom Vorgesetzten des Bearbeiters der Aktivität m bearbeitet werden, so lässt sich dies, wie in Abschnitt 5.1.3 beschrieben, häufig durch eine Serverzuordnung der Form `f(Domain(Bearb(m)))` effizient unterstützen. Dabei stellt f die Abbildung vom Domain der Angestellten einer Abteilung zum Domain ihres Vorgesetzten dar. $SV(s)$ wird (wie auch im Fall 3c), durch das Durchlaufen aller Untergebenen dieses Vorgesetzten berechnet. Dafür kann eine Bedingung der Art $Cond =$ "$Vorgesetzter = u$" verwendet werden. – Soll die Aktivität n von einem Benutzer aus derselben OE oe wie Aktivität m bearbeitet werden, der aber von diesem verschieden sein muss (4-Augen-Prinzip, Abwandlung von Fall 3c), so muss beim Durchlaufen aller Bearbeiter derjenige von Aktivität n ausgelassen werden. Die Bedingung kann hier $Cond =$ "$OE = oe \wedge Bearb \neq u$" lauten. Die $BearbZuordn_n =$ `¬ Bearb(m) ∧ Rolle ...` wird ebenso behandelt, wobei aber alle Bearbeiter mit einer passenden Rolle berücksichtigt werden müssen (außer dem von Aktivität n), und nicht nur die einer bestimmten OE. $Cond$ lautet dann "$Bearb \neq u$". (Die richtige Rolle der Bearbeiter u der Aktivität m wird in Algorithmus 5.11 Fall 3c durch die Bedingung $u \in PotBearb_m$ sichergestellt.)

5.3.2.4 Wahrscheinlichkeitsverteilung einer Workflow-Startaktivität

Instanzen der in Abschnitt 5.1.3 eingeführten Standard-WF-Typen werden (fast) ausschließlich von Bearbeitern derjenigen OE bearbeitet, in welcher sie gestartet wurden. Es werden also dieselben WF-Vorlagen in unterschiedlichen OE verwendet, wobei die WF-Instanzen vom jeweils lokalen WF-Server kontrolliert werden sollen. In ADEPT wird für diesen Sonderfall kein spezieller Serverzuordnungsausdruck verwendet. Stattdessen referenzieren alle Aktivitäten des WF in ihrer Serverzuordnung den Startknoten $Start$. Diesem ist immer die leere Aktivität zugeordnet ist. Als Bearbeiter dieses Startknotens wird der Benutzer, der diesen WF gestartet hat, vermerkt; als WF-Server der Server, auf dem die WF-Instanz gestartet wurde. Dies ist der Server im Domain dieses Benutzers. Diese beiden Daten der Startaktivität können nun in den Server- und Bearbeiterzuordnungen nachfolgender Aktivitäten referenziert werden. (Einen WF, der am Startserver verbleibt, erhält man, indem für jede Aktivität n die Serverzuordnung $ServZuordn_n =$ `Server(Start)` verwendet wird.)

Um (automatisch) ermitteln zu können, ob solche Serverzuordnungen günstig sind, müssen auch für die Startaktivität $Start$ die Bearbeiter- und Server-WV berechnet werden. Die Server-WV $ServProb_{Start}(s)$ kann aus den Domains der Benutzer abgeleitet werden, die eine Instanz dieses WF-Typs starten dürfen. Welche dies sind, wird in ADEPT durch die Bearbeiterzuordnung $BearbZuordn_{Start}$ der Startaktivität $Start$ festgelegt. Bei der Berechnung von $ServProb_{Start}(s)$ werden die Gewichte $G_{Start}(u)$ berücksichtigt, die angeben, ob der Benutzer u einen WF dieses Typs überdurchschnittlich oft bzw. selten startet.

$$\forall s\text{: } ServProb_{Start}(s) = \sum_{\substack{u \in PotBearb_{Start} \wedge \\ Domain(u)=Domain(s)}} G_{Start}(u) \;/\; \sum_{u \in PotBearb_{Start}} G_{Start}(u)$$

Für die Bearbeiter-WV gilt stets $ActorProb_{Start}(D|s) = \delta_{s,D}$, weil der Benutzer u WF-Instanzen auf seinem lokalen Server s startet. Deshalb gehören der Server s und der Benutzer u immer demselben Domain D an. Die WV $ServProb_{Start}(s)$ und $ActorProb_{Start}(D|s)$ können nun bei der Berechnung von $ServProb_n(s)$ und $ActorProb_n(D|s)$ nachfolgender Aktivitäten n verwendet werden.

5.3.3 Funktionen in Serverzuordnungen

In Serverzuordnungen kann eine Funktion f verwendet werden (Fall 4 und 5 aus Abschnitt 5.1.2). Diese Tatsache muss von den vorgestellten Algorithmen geeignet behandelt werden. Deshalb wird im Folgenden ein Verfahren vorgestellt, mit dem eine solche Funktion auch auf eine WV angewandt werden kann. Außerdem wird erklärt, wie die für eine bestimmte Serverzuordnung optimale Funktion f automatisch ermittelt werden kann.

5.3.3.1 Funktionen auf Wahrscheinlichkeitsverteilungen anwenden

Der eigentliche Zweck einer in einer Serverzuordnung verwendeten Funktion ist, einen berechneten Server auf einen anderen abzubilden. So kann die Auswertung des Ausdrucks `Server(m)` z.B. den Server s_1 ergeben. Wenn nun $f(s_1) = s_2$ festgelegt wurde, so wird in diesem Fall (bei einer Serverzuordnung `f(Server(m))`) der Server s_2 verwendet. Für einige der in dieser Arbeit vorgestellten Algorithmen ist es nun notwendig, eine solche Funktion auch auf eine WV anwenden zu können. Wie dies möglich ist, wird im Folgenden erläutert.

Um die Funktion f auf $WV(s)$ anzuwenden (eine WV, die durch einen Vektor repräsentiert wird, welcher so viele Einträge hat, wie es Domains gibt), initialisiert der Algorithmus 5.12 einen Ergebnisvektor $WV'(s)$ mit den Werten 0. Für jeden Eintrag in $WV(s)$ wird der aus der Abbildung f resultierende Server $s' = f(s)$ berechnet. Der entsprechende Eintrag aus $WV(s)$ wird dann an der Stelle s' im Ergebnis WV' berücksichtigt. Diese Vorgehensweise führt dazu, dass Wahrscheinlichkeiten, die in $WV(s)$ einem Server s zugeordnet waren, im Ergebnis WV' demjenigen Server s' zugeordnet werden, auf den $f(s)$ abbildet. Wenn die Funktion f mehrere Server auf denselben Server s' abbildet, so wird diesem die Summe der Wahrscheinlichkeiten dieser Server zugeordnet. Das ist korrekt, da er die Aufgaben aller dieser Server übernimmt.

Algorithmus 5.12 (Anwenden einer Funktion f auf eine WV $WV(s)$)
input
$WV(s)$: die ursprüngliche WV
f: die Funktion, die auf $WV(s)$ angewandt werden soll
result
$WV'(s)$: die durch Anwendung von f auf $WV(s)$ resultierende WV
begin
$\forall s : WV'(s) = 0$;
for each s **do**
$s' = f(s)$;
$WV'(s') = WV'(s') + WV(s)$;
return $WV'(s)$;
end.

5.3.3.2 Berechnung der optimalen Abbildungsfunktion f

Im Folgenden wird ein Verfahren vorgestellt, mit dem eine optimale – in einer Serverzuordnung des Typs $ServZuordn_n$ = `f(Server(m))` bzw. $ServZuordn_n$ = `f(Domain(Bearb(m)))` zu verwendende – Funktion f automatisch ermittelt werden kann. Ein solches wird benötigt, um auch

optimale Serverzuordnungen der Typen 4 und 5 automatisch ermitteln zu können. Bei der Berechnung einer solchen Funktion ist die Aktivität m, die in der festzulegenden $ServZuordn_n$ referenziert werden soll, vorgegeben. Deshalb muss jeweils eine optimale Funktion f für alle in $ServZuordn_n$ potentiell referenzierten Aktivitäten (und natürlich auch für beide oben erwähnte Formen der Serverzuordnungen) berechnet und in $PotServZuordn_n$ aufgenommen werden. Dann kann die Qualität der ermittelten Serverzuordnung (inklusive der jeweils berechneten optimalen Funktion f) mit Hilfe des Kostenmodells bewertet werden.

Wenn für die Aktivität n geprüft werden soll, ob die Verwendung der Serverzuordnung $ServZuordn_n$ = `f(Server(m))` zu einem optimalen Ergebnis führen kann, so muss erst mit Algorithmus 5.13 eine optimale Funktion f berechnet werden. Dazu werden alle Bearbeiter u_1 der referenzierten Aktivität m durchlaufen, wobei jeweils die Server-WV $DepServProb_m(s|u_1)$ ermittelt wird. Dann werden alle Bearbeiter u_2 der referenzierenden Aktivität n durchlaufen, die aufgrund von $BearbZuordn_n$ möglich sind, wenn der Benutzer u_1 die Aktivität m bearbeitet hat. Der Domain $Domain_2$ jedes Bearbeiters u_2 wird ermittelt, da dieser Domain (bzgl. u_2) für den Server der Aktivität n optimal ist. Für jedes Paar von Bearbeitern u_1 und u_2 wird die Abbildung vom Server s_1 der Aktivität m zum gewünschten Server der Aktivität n ($Domain_2$) in *Häufigkeit* vermerkt. Der Eintrag repräsentiert damit die für dieses Bearbeiterpaar optimale Abbildung der Funktion f. Er wird mit der Wahrscheinlichkeit $\frac{Gewicht_m(u_1)}{Gesamtgewicht_1} \cdot \frac{Gewicht_n(u_2)}{Gesamtgewicht_2(u_1)}$ für das Auftreten dieses Bearbeiterpaares gewichtet. Da für die Aktivität m mehrere Server s_1 möglich sind, wird der Eintrag zusätzlich mit $DepServProb_m(s_1|u_1)$ gewichtet. Wegen $\sum_s DepServProb_m(s|u_1) = 1$ wird das Gewicht des jeweiligen Bearbeiterpaares durch diese Gewichtung nicht beeinträchtigt. Nachdem alle möglichen Paare von Bearbeitern durchlaufen wurden, wird für jeden Server s_1 der Aktivität m der optimale Zielserver s_2 ausgewählt, um die Abbildungsfunktion f zu realisieren. Dies ist derjenige Server s_2 mit dem größten Eintrag in *Häufigkeit*(s_1, s_2) bei gegebenem Server s_1. Wenn derselbe maximale Eintrag in *Häufigkeit* für mehrere Server s_2 auftritt, so kann von diesen ein beliebiger ausgewählt werden, da sie alle gleich günstig sind.

Um für die Aktivität n die Eignung der Serverzuordnung $ServZuordn_n$ = `f(Domain(Bearb(m)))` zu untersuchen, kann der Algorithmus 5.13 in leicht modifizierter Form verwendet werden. Da der Server der Aktivität n dann nicht vom Server – sondern vom Bearbeiter – der Aktivität m abhängt, ist für den Algorithmus nicht die Server-WV – sondern die Bearbeiter-WV – der Aktivität m relevant. Deshalb wird für jeden Bearbeiter u_1 von Aktivität m nicht $DepServProb_m(s|u_1)$, sondern der $Domain_1 = Domain(u_1)$ ermittelt. Eben dieser Domain wird in $ServZuordn_n$ referenziert und bestimmt damit den Server der Aktivität n. Die mit $(*)$ markierte Zeile aus Algorithmus 5.13 muss deshalb durch die Folgende ersetzt werden:

$$\begin{aligned} \textit{Häufigkeit}(Domain_1, Domain_2) = \textit{Häufigkeit}(Domain_1, Domain_2) \\ + \frac{Gewicht_m(u_1)}{Gesamtgewicht_1} \cdot \frac{Gewicht_n(u_2)}{Gesamtgewicht_2(u_1)}; \end{aligned}$$

5.4 Wahrscheinlichkeitsverteilungen für Migrationen

In der Phase 1 der Verteilungsalgorithmen 4.3 - 4.5 werden die optimalen Serverzuordnungen der Aktivitäten separat betrachtet, ohne die Migrationskosten zu berücksichtigen. In der Phase 2 werden auch diese Kosten mit einbezogen. Um entscheiden zu können, ob überhaupt Kosten anfallen, muss ermittelt werden, ob zwischen zwei aufeinander folgenden Aktivitäten m und n eine Migration statt-

Algorithmus 5.13 (Ermitteln einer optimalen Abbildungsfunktion f)
input
n: die Aktivität, für deren Serverzuordnung eine optimale Funktion berechnet werden soll
m: die in $ServZuordn_n$ zu referenzierende Aktivität
$PotBearb_n$: ∀ Aktivitäten n: die Menge der potentiellen Bearbeiter der Aktivität n
$Gewicht_n(u)$: ∀ Aktivitäten n, ∀ Benutzer u: das Gewicht des Benutzers u für die Aktivität n
output
f: die optimale Funktion für $ServZuordn_n$ = `f(Server(m))`
begin
$\forall s_1, s_2$: *Häufigkeit*$(s_1, s_2) = 0$;
$Gesamtgewicht_1 = \sum_{u \in PotBearb_m} Gewicht_m(u)$;
for each $u_1 \in PotBearb_m$ **do**
$DepServProb_m(s|u_1) = DepServ(m, "Bearb = u_1", \ldots)$;
$Bearb_{u_1} = \{u \mid ActorPossible(m, u_1, n, u) = True\}$;
$Gesamtgewicht_2(u_1) = \sum_{u \in Bearb_{u_1}} Gewicht_n(u)$;
for each $u_2 \in Bearb_{u_1}$ **do**
$Domain_2 = Domain(u_2)$;
$\forall s_1$: *Häufigkeit*$(s_1, Domain_2) =$ *Häufigkeit*$(s_1, Domain_2)$
$$+ \frac{Gewicht_m(u_1)}{Gesamtgewicht_1} \cdot \frac{Gewicht_n(u_2)}{Gesamtgewicht_2(u_1)} \cdot DepServProb_m(s_1|u_1); \qquad (*)$$
for each s_1 **do**
$f(s_1) := s_2$ mit *Häufigkeit*(s_1, s_2) ist maximal;
end.

findet. Falls dem so ist, muss die Wahrscheinlichkeit $MigrProb_{m,n}(s_1, s_2)$ bestimmt werden, mit der die WF-Instanz vom Server s_1 zum Server s_2 migriert.

5.4.1 Äquivalenz von variablen Serverzuordnungen

Für die Abschätzung der zur Ausführungszeit entstehenden Migrationskosten ist es notwendig zu wissen, ob zwei Aktivitäten n und m stets vom gleichen Server kontrolliert werden ($ServZuordn_n \equiv ServZuordn_m$). Damit dies gegeben ist, müssen die Serverzuordnungsausdrücke nicht identisch sein. Außerdem bezieht sich die Gleichheit der Server auf dieselbe WF-Instanz, d.h. die Server der Aktivitäten dürfen bei unterschiedlichen WF-Instanzen verschieden sein. Im Folgenden werden Verfahren vorgestellt, mit denen die Äquivalenz von (variablen) Serverzuordnungsausdrücken festgestellt werden kann, mit denen also ermittelt wird, ob zwei Aktivitäten einer WF-Instanz stets vom selben Server kontrolliert werden.

5.4.1.1 Elementare Fälle äquivalenter Serverzuordnungen

Um zu überprüfen, ob zwei beliebige Serverzuordnungen äquivalent sind, wird diese Fragestellung zunächst für Aktivitäten gelöst, die einen direkten Bezug zueinander haben. In den nachfolgenden Abschnitten werden diese elementaren Fälle dann, unter Ausnutzung der Transitivität der Äquivalenzrelation, auf zwei beliebige Aktivitäten erweitert.

Für die Aktivitäten m und n wird derselbe Server verwendet, wenn einer der in Abb. 5.12 dargestellten elementaren Fälle vorliegt. Im Fall a wird in der Serverzuordnung der Aktivität n explizit angegeben, dass derselbe Server wie für Aktivität m verwendet werden soll und im Fall b ergibt sich dies durch

denselben Serverzuordnungsausdruck (z.B. `Domain(Bearb(l))`). Im Fall c haben die Aktivitäten k und l denselben Bearbeiter. In dessen Domain befinden sich die Server der Aktivitäten m und n, die somit identisch sind. Ein häufiger Spezialfall von c ist, dass die Aktivitäten l und m zusammenfallen, so dass sich eine Kette von Aktivitäten mit demselben Server und demselben Bearbeiter ergibt.

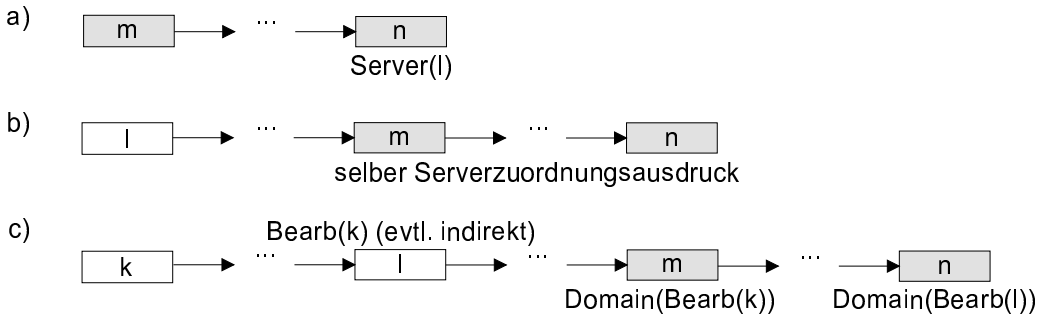


Abbildung 5.12 Fälle in denen die Aktivitäten m und n vom selben Server kontrolliert werden.

Die durch Algorithmus 5.14 definierte Funktion $SameServerDirect$ prüft für die Aktivitäten n und m, ob zwischen ihnen eine elementare Äquivalenzbeziehung besteht. Dazu wird überprüft, ob einer der in Abb. 5.12 beschriebenen Fälle vorliegt.

Algorithmus 5.14 (Elementare Äquivalenzbeziehungen: $SameServerDirect$)
input
n, m: Aktivitäten, für welche die Äquivalenz der Serverzuordnungen überprüft werden soll
$ServZuordn_n$: ∀ Aktivitäten n: die Serverzuordnung der Aktivität n
$Dep(n,m)$: ∀ Aktivitäten n, m: die Beziehung zwischen den Bearbeitern der beiden Aktivitäten
result
boolean: True, wenn zw. den Aktivitäten m und n eine elementare Äquivalenzbeziehung besteht
begin
// die Serverzuordnung der Aktivität m oder n ist noch nicht festgelegt:
if $ServZuordn_m =$ undefiniert ∨ $ServZuordn_n =$ undefiniert **then return** False;
// Fall aus Abb. 5.12a:
if $ServZuordn_m =$ `Server(n)` ∨ $ServZuordn_n =$ `Server(m)` **then return** True;
// Fall aus Abb. 5.12b:
if $ServZuordn_n = ServZuordn_m$ **then return** True;
// Fall aus Abb. 5.12c:
if $ServZuordn_m =$ `Domain(Bearb(k))` ∧ $ServZuordn_n =$ `Domain(Bearb(l))`
∧ $Dep(k,l) = B$ **then return** True;
return False;
end.

5.4.1.2 Äquivalenz beliebiger Serverzuordnungen

Soll geprüft werden, ob zwei beliebige Aktivitäten m und n vom selben Server kontrolliert werden, so muss man analysieren, ob eine „transitive Abhängigkeit“ zwischen den beiden Aktivitäten besteht. Der Algorithmus 5.15 ermittelt dazu die Menge der Aktivitäten, die vom selben Server kontrolliert werden wie die Aktivität n ($Server\_wie\_n$). Diese Menge enthält zuerst nur die Aktivität n. Dann wird in einer Schleife für jede Aktivität l geprüft, ob aufgrund der Regeln aus Abschnitt 5.4.1.1 festgestellt werden kann, dass l vom selben Server kontrolliert wird wie eine Aktivität k aus der Menge

$Server\_wie\_n$. Falls dem so ist, wird l in diese Menge aufgenommen. Dies wird solange durchgeführt, bis sich die Menge nicht mehr ändert oder die Aktivität m aufgenommen wurde. Im letzteren Fall werden die Aktivitäten m und n vom selben Server kontrolliert.

Algorithmus 5.15 (Berechnung transitiver Äquivalenzen)
input
N: die Menge der Aktivitäten der WF-Vorlage
n, m: Aktivitäten, für welche die Äquivalenz der Serverzuordnungen überprüft werden soll
$ServZuordn_n$: $\forall n \in N$: die Serverzuordnung der Aktivität n
$Dep(n, m)$: $\forall n, m \in N$: die Beziehung zwischen den Bearbeitern der beiden Aktivitäten
result
boolean: True, wenn die Aktivitäten m und n stets vom selben Server kontrolliert werden
begin
$Server\_wie\_n = \{n\}$;
do
$changed =$ False;
for each $l \in N$ **do**
for each $k \in Server\_wie\_n$ **do**
if $SameServerDirect(l, k, ServZuordn_*, Dep(*, *))$ **then**
$Server\_wie\_n = Server\_wie\_n \cup \{l\}$;
$changed =$ True;
while $changed = True \wedge m \notin Server\_wie\_n$;
return $m \in Server\_wie\_n$;
end.

5.4.1.3 Äquivalenzklassen von Serverzuordnungen

Den (naiven) Algorithmus 5.15 jedes Mal auszuführen, wenn die Äquivalenz der Serverzuordnungen zweier Aktivitäten getestet werden soll, bedeutet einen sehr großen Aufwand. Eine effiziente Alternative stellt die einmalige Berechnung von *Serverklassen* dar. Um die Äquivalenz der Serverzuordnungen der Aktivitäten m und n festzustellen, muss nach deren Berechnung nur noch geprüft werden, ob sie derselben Serverklasse angehören:
$ServZuordn_m \equiv ServZuordn_n \quad \Leftrightarrow \quad ServerKlasse_m = ServerKlasse_n$

Zur Berechnung der Serverklasse $ServerKlasse_n$ einer Aktivität n testet der Algorithmus 5.16 für jede Aktivität n, ob sie in eine der schon ermittelten Serverklassen fällt. Dazu wird mit Algorithmus 5.14 für alle Aktivitäten m, deren Serverklasse schon feststeht, geprüft, ob die Aktivitäten m und n vom selben Server kontrolliert werden. Falls dem so ist, wurde die Serverklasse von Aktivität n gefunden und die Suche kann beendet werden. Wurde keine Aktivität m gefunden, die diese Bedingung erfüllt, so begründet Aktivität n eine neue Serverklasse.

5.4.2 Migrationswahrscheinlichkeiten

Die Wahrscheinlichkeiten für Migrationen beim Übergang von Aktivität m nach Aktivität n können in Form einer *Migrationsmatrix* angegeben werden. Dabei gibt ein Eintrag an, wie groß die bedingte Wahrscheinlichkeit ist, dass zum Server s_2 migriert werden muss, wenn die Aktivität m vom Server s_1 kontrolliert wird. Es gilt also: $DepMigrProb_{m,n}(s_2|s_1)$ = *P(Server* s_2 *kontrolliert Aktivität* n | *Server* s_1 *kontrolliert Aktivität* m*)*

Algorithmus 5.16 (Berechnung der Serverklassen)
input
N: die Menge der Aktivitäten der WF-Vorlage
$ServZuordn_n$: $\forall n \in N$: die Serverzuordnung der Aktivität n
$Dep(n,m)$: $\forall n,m \in N$: die Beziehung zwischen den Bearbeitern der beiden Aktivitäten
output
$ServerKlasse_n$: $\forall n \in N$: die Server-Äquivalenzklasse der Aktivität n
begin
$maxKlasse = 0$;
$\forall n \in N$: $ServerKlasse_n = NULL$;
for each Aktivität $n \in N$ **do**
for each Aktivität $m \in N$ mit $ServerKlasse_m \neq NULL$ **do**
if $SameServerDirect(m,n,ServZuordn_*, Dep(*,*))$ **then**
$ServerKlasse_n = ServerKlasse_m$;
exit loop;
if $ServerKlasse_n = NULL$ **then**
$maxKlasse = maxKlasse + 1$;
$ServerKlasse_n = maxKlasse$;
end.

Damit ergibt sich die vom Kostenmodell aus Abschnitt 4.3 verwendete Migrationswahrscheinlichkeit $MigrProb$ als:
$MigrProb_{m,n}(s_1,s_2) = ServProb_m(s_1) \cdot DepMigrProb_{m,n}(s_2|s_1)$

In den nun folgenden Unterabschnitten wird beschrieben, wie die WV $DepMigrProb$ ermittelt werden kann.

5.4.2.1 Einfache Fälle

Werden die Aktivitäten m und n vom selben Server kontrolliert ($ServerKlasse_m = ServerKlasse_n$), so muss nie migriert werden, so dass sich $DepMigrProb_{m,n}(s_2|s_1) = \delta_{s_1,s_2}$ ergibt.

Sind die Serverzuordnungen der Aktivitäten m und n nicht äquivalent, so bietet die Nutzung der Server-WV $ServProb_n(s)$ eine einfache Möglichkeit zur Approximation der Migrationswahrscheinlichkeiten. Die Server-WV gibt an, mit welcher Wahrscheinlichkeit sich die Instanz nach der Migration zur Aktivität n am Server s befindet. Die Migrationswahrscheinlichkeiten ergeben sich damit als:
$\forall s_1, s_2 : DepMigrProb_{m,n}(s_2|s_1) = ServProb_n(s_2)$

5.4.2.2 Abhängigkeiten zwischen Aktivitäten

Bei der im vorherigen Anschnitt beschriebenen Approximation wird die Annahme vorausgesetzt, dass die Server der Aktivitäten m und n unabhängig voneinander gewählt werden (wenn sie nicht äquivalent sind). Das ist jedoch nicht der Fall, wenn z.B. die Server- und Bearbeiterzuordnungen der beiden Aktivitäten jeweils von Bearbeitern derselben OE abhängen. Deshalb ist es günstiger, zur Berechnung der Migrationswahrscheinlichkeiten von der Aktivität m zur Aktivität n den Algorithmus 5.17 zu verwenden, falls eine Abhängigkeit zwischen den Bearbeitern der Aktivitäten besteht ($Dep(n,m) \neq NULL$). Dieser Algorithmus berücksichtigt die Abhängigkeiten zwischen den Bearbeitern der beiden Aktivitäten.

Der Algorithmus 5.17 berechnet die tatsächlichen Migrationswahrscheinlichkeiten, da er alle aufgrund von $BearbZuordn_n$ erlaubten Kombinationen von Bearbeitern u_1 und u_2 für die Aktivitäten m und n durchläuft, die Wahrscheinlichkeit ihres Auftretens berechnet und die bei diesem Paar auftretende Migration in $DepMigrProb$ berücksichtigt. Die Wahrscheinlichkeit, dass der Benutzer u_1 die Aktivität m bearbeitet, beträgt $Gewicht_m(u_1)/Gesamtgewicht_1$. Die bedingte Wahrscheinlichkeit, dass Benutzer u_2 unter dieser Voraussetzung Aktivität n bearbeitet, beträgt $Gewicht_n(u_2)/Gesamtgewicht_2(u_1)$. Mit diesen Wahrscheinlichkeiten gewichtet geht die Migration in $DepMigrProb$ ein. Ergibt die Berechnung von $DepServProb_m(s|u_1)$ bzw. $DepServProb_n(s|u_2)$ für das betrachtete Bearbeiterpaar u_1 und u_2, dass mehrere Server für die Kontrolle der Aktivitäten in Frage kommen, so werden die Bearbeiter anteilsmäßig bei jedem dieser Server berücksichtigt. Dadurch ergibt sich der Faktor $DepServProb_m(s_1|u_1) \cdot DepServProb_n(s_2|u_2)$. Durch die abschließende Normalisierung wird $DepMigrProb$ noch in eine Form gebracht, bei der die Summe jeder Zeile 1 ergibt. Damit stellt die Zeile s_1 die bedingte Wahrscheinlichkeit dar, mit der zum Server s_2 migriert werden muss, wenn die Aktivität m vom Server s_1 kontrolliert wird. Zeilen der Form $(0 \ldots 0)$ ergeben sich, wenn der Server s_1 die Aktivität m niemals kontrolliert und sind deshalb nicht relevant ($ServProb_m(s_1) = 0 \Rightarrow MigrProb_{m,n}(s_1, s_2) = ServProb_m(s_1) \cdot DepMigrProb_{m,n}(s_2|s_1) = 0$). Sie können deshalb bei der Normalisierung mit beliebigen Werten belegt werden. Im Anhang B.2 wird bewiesen, dass im Algorithmus 5.17 alle Fälle berücksichtigt werden, und dass die Faktoren korrekt gewählt sind.

Algorithmus 5.17 (Migrationswahrscheinlichkeiten (keine Unabhängigkeit))
input
m: die Quellaktivität der Migration
n: die Zielaktivität der Migration
$PotBearb_n$: $\forall$ Aktivitäten n: die Menge der potentiellen Bearbeiter der Aktivität n
$Gewicht_n(u)$: $\forall$ Aktivitäten n, $\forall$ Benutzer u: das Gewicht des Benutzers u für die Aktivität n
output
$DepMigrProb_{m,n}(s_2|s_1)$: die abhängige Migrations-WV
begin
$\forall s_1, s_2$: $DepMigrProb_{m,n}(s_2|s_1) = 0$;
$Gesamtgewicht_1 = \sum_{u \in PotBearb_m} Gewicht_m(u)$;
for each $u_1 \in PotBearb_m$ **do**
$DepServProb_m(s|u_1) = DepServ(s, "Bearb = u_1", \ldots)$;
$Bearb_{u_1} = \{u \mid ActorPossible(m, u_1, n, u, PotBearb_n) = True\}$; // siehe Algorithmus 5.3
$Gesamtgewicht_2(u_1) = \sum_{u \in Bearb_{u_1}} Gewicht_n(u)$;
for each $u_2 \in Bearb_{u_1}$ **do**
$DepServProb_n(s|u_2) = DepServ(n, "Bearb = u_2", \ldots)$;
for each s_1, s_2 **do**
$DepMigrProb_{m,n}(s_2|s_1) = DepMigrProb_{m,n}(s_2|s_1)$
$+ \frac{Gewicht_m(u_1)}{Gesamtgewicht_1} \cdot \frac{Gewicht_n(u_2)}{Gesamtgewicht_2(u_1)} \cdot DepServProb_m(s_1|u_1) \cdot DepServProb_n(s_2|u_2)$;
normalisiere jede Zeile von $DepMigrProb_{m,n}(s_2|s_1)$ so, dass $\forall s_1$: $\sum_{s_2} DepMigrProb_{m,n}(s_2|s_1) = 1$;
end.

Der mit Algorithmus 5.17 behandelte Fall ist in der Praxis sehr relevant, da WF-Instanzen häufig für die Dauer mehrerer Aktivitäten innerhalb derselben OE verbleiben. Außerdem bilden die Benutzer einer OE oft exakt einen Domain. Dies gelte für das in Abb. 5.13 dargestellte Beispiel. Bei diesem sind die Serverzuordnungen der Aktivitäten m und n nicht äquivalent (vgl. Abschnitt 5.4.1). Wird angenommen, dass für diese Aktivitäten zwei Server – jeweils mit der Wahrscheinlichkeit 0,5 – verwendet werden, so ergibt sich durch die einfache Approximation vom $DepMigrProb_{m,n}(s_2|s_1)$

durch $ServProb_n(s_2)$ nach Abschnitt 5.4.2.1 eine Migrationswahrscheinlichkeit von 50%. In Wirklichkeit tritt aber keine Migration auf, weil die Bearbeiter der Aktivitäten l und m (deren Domains die Server von m und n determinieren) derselben OE und damit demselben Domain angehören. Der Algorithmus 5.17 berücksichtigt dies, da bei allen Kombinationen von Bearbeitern der Aktivitäten m und n jeweils derselbe Server (nämlich der in der OE dieser Bearbeiter) ermittelt wird, womit sich $DepMigrProb_{m,n}(s_2|s_1) = \delta_{s_1,s_2}$ ergibt. Befindet sich lediglich der Großteil der Benutzer einer OE im gleichen Domain, so ergibt sich annähernd $DepMigrProb_{m,n}(s_2|s_1) = \delta_{s_1,s_2}$. In beiden Fällen wird erkannt, dass die Migrationskosten wesentlich niedriger sind, als wenn Unabhängigkeit angenommen worden wäre.

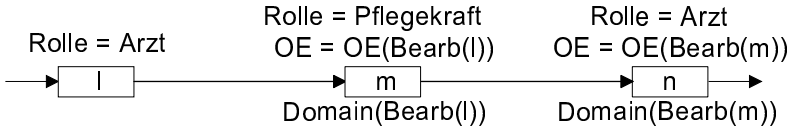


Abbildung 5.13 Beispiel für abhängige der Migrationswahrscheinlichkeiten.

5.5 Zusammenfassung und Ausblick

In den Kapiteln 3 und 4 wurde beschrieben, wie in ADEPT$_{distribution}$ die verteilte WF-Steuerung realisiert wird und wie die dafür benötigten Serverzuordnungen der Aktivitäten systemunterstützt ermittelt werden können. Die vorgestellten Verfahren stoßen aber an ihre Grenzen, wenn eine WF-Vorlage einen großen Anteil abhängiger Bearbeiterzuordnungen enthält. Um auch für solche Szenarien eine bzgl. der Kommunikationskosten effiziente verteilte WF-Steuerung zu ermöglichen, wurde im vorliegenden Kapitel das Konzept der variablen Serverzuordnungen eingeführt. Dessen Vorteile wurden an einigen Beispielen demonstriert (eine quantitative Analyse folgt in Kapitel 9) und es wurde geklärt, wie variable Serverzuordnungen zur Ausführungszeit realisiert werden können. Die weitaus größte Herausforderung war aber, ein Verfahren zu entwickeln, mit dem geeignete variable Serverzuordnungen zur Modellierungszeit automatisch ermittelt werden können. Dabei konnte zwar weiterhin das Kostenmodell aus Kapitel 4 verwendet werden, doch die Berechnung der dafür benötigten WV gestaltet sich wesentlich schwieriger. Dennoch ist es gelungen, Algorithmen zu entwickeln, mit denen diese WV berechnet werden können.

Im Kontext von variablen Serverzuordnungen lassen sich noch weitere interessante Fragestellungen identifizieren. Auf diese wird in [BD00b] ausführlich eingegangen, wegen ihrer beschränkten Relevanz für die Praxis werden sie hier nur kurz angesprochen: In Abschnitt 4.6 wurde – für unabhängige Bearbeiterzuordnungen – ein Verfahren zur Berechnung einer geeigneten Verteilung der Benutzer auf die Teilnetze vorgestellt. Das Problem ist nun, dass dieses Verfahren bei abhängigen Bearbeiterzuordnungen nicht ohne Erweiterungen eingesetzt werden kann, falls Instanzen eines WF-Typs in unterschiedlichen OE bearbeitet werden. Vereinfacht dargestellt lässt sich dieses Problem dadurch lösen, dass zuerst Benutzern verschiedener OE unterschiedliche Domains zugeordnet werden. Dies ist sicherlich sinnvoll, weil diese Benutzer unterschiedliche WF-Instanzen bearbeiten und deshalb keinen Bezug zueinander haben. Umfasst eine OE zu viele Benutzer, um sie im selben Teilnetz zu platzieren, so wird auf diese Bearbeiter das Verfahren aus Abschnitt 4.6 angewendet, so dass Benutzer (derselben OE) mit einer hohen Korrelation der Aufgaben ermittelt werden.

Bei manchen Anwendungen sind auch abhängige Bearbeiterzuordnungen denkbar, welche mehr als eine Aktivität referenzieren. Ein Beispiel für eine solche *zusammengesetzte abhängige Bear-*

beiterzuordnung ist in Abb. 5.14a für die Aktivität „Befund erstellen“ dargestellt. Häufig deutet die Verwendung einer zusammengesetzten Bearbeiterzuordnung allerdings auf eine unsaubere Modellierung hin. So kann auch die in Abb. 5.14a verwendete zusammengesetzte Bearbeiterzuordnung durch die in Abb. 5.14b dargestellte Modellierung vermieden werden. In [BD00b] werden Verfahren vorgestellt, mit denen bei Präsenz von zusammengesetzten abhängigen Bearbeiterzuordnungen optimale (ebenfalls zusammengesetzte) variable Serverzuordnungen berechnet werden können (z.B. `Zweig 1 gewählt: Domain(Bearb(Röntgen)), Zweig 2 gewählt: Domain(Bearb(Sonographie))`). Für den extrem seltenen Fall einer tatsächlich benötigten zusammengesetzten abhängigen Bearbeiterzuordnung kann eine solche Serverzuordnung aber auch explizit durch den Modellierer vorgegeben werden (als Serverzuordnung vom Typ 6).

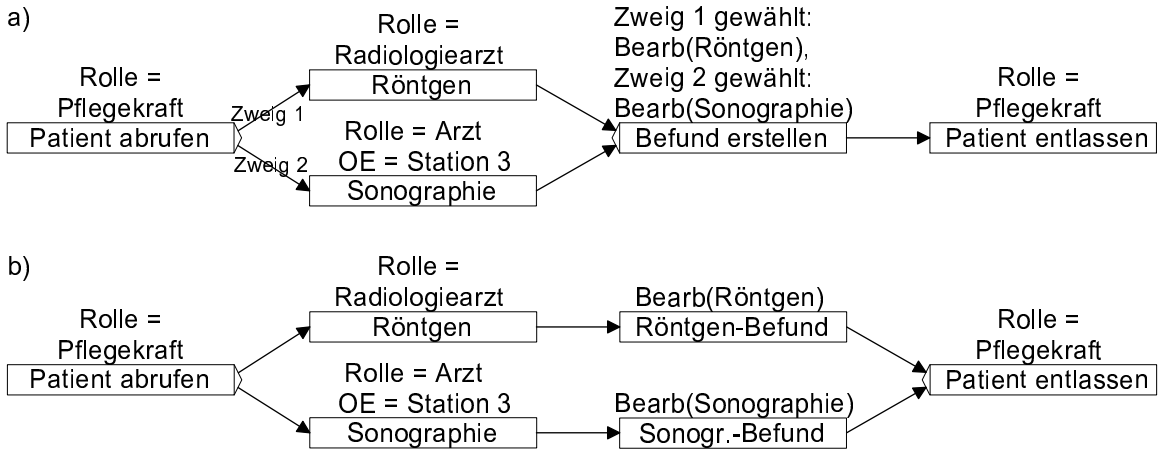


Abbildung 5.14 Beispiel für eine (vermeidbare) zusammengesetzte abhängige Bearbeiterzuordnung.

Kapitel 6

Optimierung von Migrationen

Mit den bisher in dieser Arbeit vorgestellten Verfahren lassen sich diejenigen (variablen) Serverzuordnungen ermitteln, die zu einer bzgl. der Kommunikationskosten optimalen Ausführung der WF-Instanzen führen. Eine WF-Instanz wird dann abschnittsweise von verschiedenen WF-Servern kontrolliert, weshalb Migrationen notwendig werden. Bei diesen besteht ein großes Optimierungspotential, da die Durchführung einer Migration i.d.R. die Übertragung einer großen Datenmenge erfordert, und da bei Migrationen häufig nicht benötigte Daten übertragen werden. So kann der Zielserver einer Migration bereits früher einmal an der entsprechenden WF-Instanz beteiligt gewesen sein, so dass ihm gewisse Instanzdaten schon bekannt sind. In diesem Fall ist es unnötig, bereits vorhandene Daten nochmals zu übertragen. Im vorliegenden Kapitel werden zur Ausführungszeit der WF-Instanzen zur Anwendung kommende Verfahren vorgestellt, die solche und ähnliche Fälle ausschließen. Wie in [BRD01b] durch ein Zahlenbeispiel eindrucksvoll gezeigt wurde, ist es hierdurch möglich, das Kommunikationsaufkommen bei Migrationen drastisch zu reduzieren. Die Ausnutzung des existierenden Optimierungspotentials ist insbesondere für unternehmensweite und -übergreifende Anwendungen unverzichtbar, da bei den entsprechenden Systemen die Kommunikation häufig (zumindest teilweise) über ein WAN oder eine sonstige langsame Verbindung (Internet, ISDN) abgewickelt wird. Dennoch gibt es unseres Wissens bisher keine Arbeiten, die systematisch untersuchen, wie Migrationen effizient realisiert werden können.

Im nachfolgenden Abschnitt werden die Problemstellungen detailliert untersucht, die sich im Zusammenhang mit der effizienten Realisierung von Migrationen ergeben. Nachdem schon in Abschnitt 3.3.3 festgelegt wurde, auf welche Art und Weise der interne Zustand einer WF-Instanz migriert werden soll, werden im Abschnitt 6.2 Optimierungen dieses Verfahrens diskutiert. Die Migration von Datenelementen der WF-Instanz wird in Abschnitt 6.3 betrachtet. Das Kapitel endet mit einer Zusammenfassung der gewonnenen Erkenntnisse in Abschnitt 6.4.

6.1 Problemstellung

Bei Migrationen von WF-Instanzdaten zwischen WF-Servern entstehen hohe Kosten. Deshalb sollen Verfahren entwickelt werden, mit denen das dabei zu übertragende Datenvolumen reduziert werden kann. Während wir uns bisher auf Verfahren zur Reduzierung der Kommunikationslast konzentriert haben, die auf der Vorberechnung von geeigneten Serverzuordnungen zur *Modellierungszeit* basieren, werden bei den in diesem Kapitel vorgestellten Verfahren die Kommunikationskosten durch zur

Ausführungszeit durchgeführte Optimierungen weiter reduziert. Dies wird erreicht, indem nicht alle für die betroffene WF-Instanz verfügbaren Daten zum Migrationszielserver übertragen werden, sondern nur diejenigen, die dieser Server tatsächlich für die korrekte WF-Ausführung benötigt und die er noch nicht besitzt. Ein WF-Server kann beispielsweise bereits über Daten verfügen, wenn er früher an der Kontrolle der WF-Instanz beteiligt war, oder wenn ihm bestimmte Informationen bei vorausgehenden Migrationen übermittelt worden sind. Da die Migrationen durch entsprechende Maßnahmen „billiger“ werden, können sie ggf. häufiger und damit auch für kleinere Prozessfragmente durchgeführt werden. Die WF-Steuerung erfolgt dementsprechend meist im Teilnetz derjenigen Benutzer, welche die Aktivität bearbeiten, so dass sich die Antwortzeiten und die Verfügbarkeit des WfMS verbessern (vgl. Abschnitt 3.2.1).

Um einen Eindruck zu vermitteln, welche Art von Optimierungen im Zusammenhang mit der Durchführung von Prozessmigrationen im Einzelnen möglich sind, wollen wir anhand von Abb. 6.1 einige Beispiele betrachten:

- Die Zustandsinformation (inkl. Ablaufhistorie) zu den Aktivitäten a und b wird bei herkömmlichen Migrationen über beide parallele Zweige zur AND-Join-Aktivität g transportiert. Das heißt, sie würde sowohl bei der Migration $M_{d,g}$ als auch bei $M_{f,g}$ zum Server s_5 übertragen werden. Wie man leicht sieht, kann eine dieser beiden (redundanten) Übertragungen eingespart werden.
- Die Aktivität c schreibt einen Ausgabeparameter in das Datenelement d_1, auf das von der Aktivität g lesend zugegriffen wird (nicht aber von d). Dann ist es wenig sinnvoll, das Datenelement entlang des Kontrollflusses vom Server s_2 über den Server s_3 zum Server s_5 zu übertragen. Eine Optimierung wäre es, das Datenelement direkt vom Server s_2 zum Server s_5 zu übertragen. Dies ist besonders rentabel, wenn sehr große Datenelemente (z.B. Multimediadaten wie Videos, Bitmaps, ...) betroffen sind.
- Von der Aktivität g wird das Datenelement d_2 benötigt, das von der Aktivität a geschrieben und von c und d gelesen wurde. Der Wert dieser Variablen kann von den WF-Servern s_1, s_2 und s_3 bezogen werden. Es existiert insofern Optimierungspotential, als dass das Datenelement von demjenigen Server bezogen werden sollte, zu dem die leistungsfähigste Kommunikationsverbindung besteht. In jedem Fall sollte eine redundante Übertragung vermieden werden. Angenommen die Server s_1 und s_3 sind mit dem Server s_5 nur über eine ISDN-Verbindung verbunden. Der Server s_2 liege aber in einem zum Server s_5 benachbarten Teilnetz mit einer Verbindung über ein schnelles Gateway. Dann ist es wesentlich günstiger, wenn das (große) Datenelement vom Server s_2 bezogen wird.

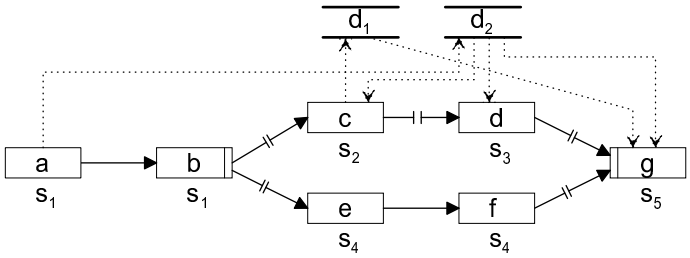


Abbildung 6.1 WF mit Optimierungspotential bei Migrationen.

Die Beispiele zeigen, dass bei „naiver“ Realisierung Daten unnötigerweise zu einer Partition übertragen werden können. Eine solch redundante Übertragung sollte auf alle Fälle vermieden werden. Im Folgenden werden Verfahren vorgestellt, mit denen Migrationen effizient durchgeführt werden können, indem die entstehenden Migrationskosten minimiert werden. Dazu werden im Einzelnen

Verfahren zur optimalen und redundanzfreien Migration der Kontrolldaten einer WF-Instanz und zur Übertragung von Datenelementen vorgestellt.

6.2 Optimierte Übertragung von Workflow-Kontrolldaten

Wie in Abschnitt 3.3.3.1 ausführlich erörtert wurde, ist die günstigste Methode, um den Zustand einer WF-Instanz zu migrieren, die entsprechende Ablaufhistorie zu übertragen. Bei einer Übertragung der vollständigen Ablaufhistorie kann es allerdings zu redundanten Datenübertragungen kommen. So werden bei dem in Abb. 6.1 dargestellten Beispiel die Ablaufhistorieneinträge der Aktivitäten a und b bei den Migrationen $M_{d,g}$ und $M_{f,g}$ zum Server s_5 übertragen. Im dem nun folgenden Abschnitt werden verbesserte Verfahren vorgestellt, bei denen zum Zielserver nur die wirklich noch benötigten Historieneinträge übertragen werden.

6.2.1 Versenden von Ablaufhistorieneinträgen

Die nahe liegendste Idee zur Reduzierung des Datenvolumens beim Übertragen von WF-Kontrolldaten besteht darin, nur den benötigten Ausschnitt der Ablaufhistorie und nicht die komplette Ablaufhistorie an den Migrationszielserver zu senden. Das entsprechende Verfahren ist in [Zei99] ausführlich und formal beschrieben. Aus Platzgründen werden hier nur das Grundprinzip erläutert und die entstehenden Nachteile diskutiert. In Abschnitt 6.2.2 wird dann ein verbessertes Verfahren vorgestellt, das diese Nachteile vermeidet.

Um bei einer Migration nicht immer die gesamte Ablaufhistorie einer WF-Instanz übertragen zu müssen, berechnet der Quellserver, welcher Teil dieser Historie vom Zielserver für die weitere Ausführung noch benötigt wird. Dementsprechend wird nur dieser Ausschnitt bei der Migration übertragen. Dazu sucht der Quellserver in der Historie nach Einträgen, deren zugehörige Aktivitäteninstanz vom Zielserver kontrolliert wurde (sofern vorhanden). Bei der Übertragung der Ablaufhistorie können diese Einträge dann weggelassen werden. Dasselbe gilt für Historieneinträge, die sich auf Vorgängeraktivitäteninstanzen (bzgl. des Kontrollflusses) dieser Aktivitäteninstanzen beziehen, da auch diese Historieneinträge dem Zielserver schon bekannt sind. Der Grund dafür ist, dass ein WF-Server alle Historieneinträge kennt, die zu Vorgängeraktivitäten der von ihm kontrollierten Aktivitäteninstanzen gehören (vgl. Abschnitt 3.3.3.2).

Durch dieses Verfahren werden Historieneinträge, von denen der Quellserver einer Migration sicher weiß, dass sie auf dem Zielserver schon bekannt sind, an diesen nicht mehr übertragen. So werden bei dem in Abb. 6.2 dargestellten Beispiel einer parallelen Verzweigung die Historieneinträge zu den Aktivitäteninstanzen a und b weder bei der Migration $M_{f,g}$ noch bei $M_{h,i}$ an den Server s_2 übertragen, da dieser bereits die Aktivität b kontrolliert hat und deshalb die zu Aktivität b und zur Vorgängeraktivität a gehörenden Historieneinträge schon kennt.

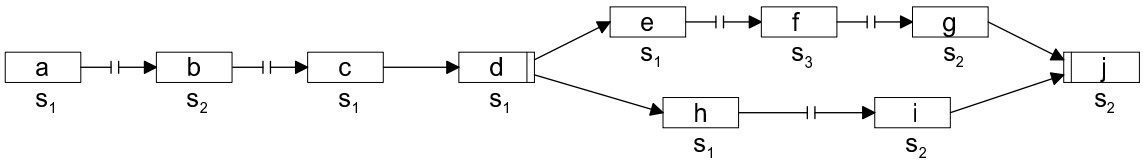


Abbildung 6.2 Beispiel für die Migration von WF-Kontrolldaten.

Bei dem oben skizzierten Verfahren kann es aber immer noch zur redundanten Übertragung von Historieninformation kommen. Das Problem des Verfahrens ist, dass der Quellserver einer Migration nicht exakt weiß, welche Historieneinträge beim Zielserver schon vorhanden sind. Nehmen wir in dem Beispiel aus Abb. 6.2 an, dass die Migration $M_{f,g}$ vor $M_{h,i}$ ausgeführt wird. Dann müssen bei der Migration $M_{f,g}$ die Historieneinträge der Aktivitäten c und d übertragen werden. Bei der später ausgeführten Migration $M_{h,i}$ kann der Quellserver s_1 aber nicht wissen, dass die entsprechenden Einträge dem Zielserver s_2 schon bekannt sind. Diese nicht vorhandene Information führt dazu, dass er die Einträge unnötigerweise überträgt. Das vorgestellte Verfahren ermöglicht also zwar eine Reduzierung der zu übertragenden Datenmenge, redundante Datenübertragung lässt sich damit aber nicht völlig ausschließen.

6.2.2 Anfordern von Ablaufhistorieneinträgen

Die redundante Übertragung von Ablaufhistorieneinträgen kann ausgeschlossen werden, wenn der Zielserver einer Migration dem Quellserver Information darüber liefert, welche Historieneinträge ihm bereits bekannt sind.[1] Ein Verfahren hierfür, wird nun vorgestellt. Dabei kann eine redundante Übertragung von Historieneinträgen nur dann ausgeschlossen werden, wenn für eine WF-Instanz nie mehrere Migrationen zeitlich überlappend bei demselben WF-Server eingehen. Ansonsten kann der Zielserver nicht wissen, welche Historieneinträge noch durch die anderen gleichzeitig ablaufenden Migrationen geliefert werden. Deshalb verwendet der Zielserver einer Migration Sperren, um sicherzustellen, dass zu jedem Zeitpunkt für eine WF-Instanz maximal eine Migration eingehen kann, d.h. er blockiert ggf. kurzfristig die weitere Bearbeitung einer eingehenden Migration.

Das im Folgenden vorgestellte Verfahren basiert auf der Idee, dass der Zielserver der durchzuführenden Migration beim Quellserver diejenigen Teile der Ablaufhistorie anfordert, die bei ihm noch nicht vorhanden sind. Das Verfahren gliedert sich in 3 Phasen:

1. Der Quellserver informiert den Zielserver über die anstehende Migration. Dazu überträgt er ihm außer der ID der zu migrierenden WF-Instanz jeweils die ID der Quell- und Zielaktivitäten der Migration. Für das Beispiel der Migration $M_{f,g}$ aus Abb. 6.2 ergibt sich also die Übertragung: <WF-Inst-ID, f, g >
2. Der Zielserver der Migration fordert nun beim Quellserver den noch fehlenden Teil der Ablaufhistorie an. Dazu muss er ihm mitteilen, über welche Historieneinträge er bereits verfügt. Die einfachste Lösung, die IDs aller zugehörigen Aktivitäteninstanzen zu übertragen, wäre aber unnötig aufwendig. Sind die Historieneinträge einer Aktivitäteninstanz dem Zielserver bekannt, so kennt er nämlich auch die Einträge zu Vorgängeraktivitäten dieser Aktivitäteninstanz. Deshalb genügt es, die Menge $LastActivities$ zu übertragen, welche die IDs derjenigen Aktivitäteninstanzen enthält, von denen dem Zielserver keine Historieneinträge zu Nachfolgeraktivitäten bekannt sind. Dies sind sozusagen die „letzten dem Zielserver bekannten Aktivitäten“ der WF-Instanz. Es kann vorkommen, dass die Menge $LastActivities$ mehrere Aktivitäten enthält, da der Zielserver einer Migration Historieneinträge zu Aktivitäten besitzen kann, die verschiedenen parallelen Zweigen angehören. Dann muss die jeweils letzte bekannte Aktivität jedes Zweiges in $LastActivities$ enthalten sein.

   Mit dem Algorithmus 6.1 kann der Zielserver einer Migration die Menge $LastActivities$ be-

[1] Eine Alternative dazu wäre, dass die anderen Server des WfMS Information darüber austauschen, welche Historieneinträge schon zu diesem Zielserver migriert wurden. Da ein solches Verfahren aber sehr komplex wäre und außerdem nicht weniger Kommunikation erfordern würde, wird dieser Ansatz nicht weiter verfolgt.

rechnen. Dazu reduziert er die lokal bei ihm vorhandene Ablaufhistorie $OldHistory$ der WF-Instanz auf die Vorgängeraktivitäten der Migrationsquellaktivität, da nur dieser Teil der WF-Instanz für die Migration relevant ist. Dadurch ergibt sich die Ablaufhistorie $RelevantHistory$. Der hinterste Eintrag L von $RelevantHistory$ gehört zur einer Aktivitäteninstanz l, deren Historieneinträge am Zielserver bekannt sind. Da der Zielserver auch über die Historieneinträge aller Vorgängeraktivitäten von l verfügt, entfernt er diese aus der Ablaufhistorie $RelevantHistory$. Die Aktivitäteninstanz l wird in $LastActivities$ aufgenommen, weil die Vorgängeraktivitäten von l (inkl. l) am Zielserver bekannt sind, nicht aber die Nachfolgeraktivitäten. Der gesamte Vorgang wird solange wiederholt, bis in der Historie $RelevantHistory$ keine Einträge mehr existieren. Ist dies erreicht, so wurde die vollständige Menge $LastActivities$ ermittelt. Im Beispiel der Migration $M_{f,g}$ aus Abb. 6.2 ergibt sich $LastActivities = \{b\}$, weil der Zielserver (Server s_2) nur über Information zu den Aktivitäten a und b verfügt (b wurde vom Server s_2 kontrolliert). Da a eine Vorgängeraktivität von b ist, wird a nicht in $LastActivities$ aufgenommen.

Algorithmus 6.1 (Zielserver: Anfordern von Ablaufhistorieneinträgen)
input
$SourceAct$: Quellaktivitäteninstanz der Migration
$OldHistory$: am Zielserver bekannter Teil der Ablaufhistorie
output
$LastActivities$: Menge von Aktivitäteninstanzen, bis zu denen die Ablaufhistorie dem Zielserver bekannt ist
begin
$LastActivities = \emptyset$;
// für Migration sind nur Vorgängeraktivitäteninstanzen der Quellaktivität (inklusive) relevant
// ($\cap$ und $-$ für Historien wurden in Abschnitt 2.2.3.4 definiert)
$RelevantHistory$
$= OldHistory \cap (\{SourceAct\} \cup PredInst^*_{CONTROL\_E,SYNC\_E,LOOP\_E}(SourceAct))$;
while $RelevantHistory \neq \emptyset$ **do**
// L ist der hinterste Eintrag der Historie $RelevantHistory$, l die zugehörige Aktivitäteninstanz
$L = LastEntry(RelevantHistory)$;
$l = ActivityInstance(L)$;
$LastActivities = LastActivities \cup \{l\}$;
// entferne Einträge zu l und aller Vorgänger bzgl. Kontrollfluss aus Ablaufhistorie
$RelevantHistory = RelevantHistory - (\{l\} \cup PredInst^*_{CONTROL\_E,SYNC\_E,LOOP\_E}(l))$;
end.

3. Nachdem der Quellserver der Migration die Menge $LastActivities$ empfangen hat, kann er mit Algorithmus 6.2 die zu übertragende Ablaufhistorie $MigrHistory$ berechnen. Dazu reduziert er die bei ihm vorhandene Ablaufhistorie $LocalHistory$ auf Vorgänger der Quellaktivität der Migration (wie der Zielserver in Algorithmus 6.1). Anschließend entfernt er alle Einträge zu Aktivitäteninstanzen $l \in LastActivities$ und zu deren Vorgängern aus $MigrHistory$, so dass sich die zu übertragende Ablaufhistorie ergibt. Betrachten wir nochmals das oben angesprochene Beispiel der Migration $M_{f,g}$, bei dem in Schritt 2 die Menge $LastActivities = \{b\}$ ermittelt wurde. In Schritt 3 werden dann die Einträge zu den Aktivitäteninstanzen a und b aus der lokal vorhandenen Historie (mit Einträgen zu den Aktivitäten a, b, c, d, e, f) entfernt, so dass die Historieneinträge der Aktivitäten c, d, e, f übertragen werden.

Wird nach der oben beschriebenen Migration $M_{f,g}$ die Migration $M_{h,i}$ zum selben Zielserver s_2 durchgeführt,[2] so kennt dieser Server die Einträge zu den Aktivitäten a, b, c, d, e und f bereits.

[2]Wie zu Beginn dieses Abschnitts erläutert wurde, ist die überlappende Ausführung von Migrationen ausgeschlossen.

Algorithmus 6.2 (Quellserver: Anfordern von Ablaufhistorieneinträgen)
input
$SourceAct$: Quellaktivitäteninstanz der Migration
$LastActivities$: Menge der letzten am Zielserver bekannten Aktivitäteninstanzen
$LocalHistory$: am Quellserver vorhandene Ablaufhistorie
output
$MigrHistory$: der bei der Migration zu übertragende Teil der Ablaufhistorie
begin
// für Migration sind nur Vorgängeraktivitäten der Quellaktivität relevant
$MigrHistory$
$= LocalHistory \cap (\{SourceAct\} \cup PredInst^{*}_{CONTROL\_E,SYNC\_E,LOOP\_E}(SourceAct))$;
// Einträge zu bekannten Aktivitäteninstanzen aus Ablaufhistorie entfernen
for each $l \in LastActivities$ **do**
// Einträge zu l und Vorgängern von l aus Historie streichen
$MigrHistory = MigrHistory - (\{l\} \cup PredInst^{*}_{CONTROL\_E,SYNC\_E,LOOP\_E}(l))$;
end.

Durch Anwendung von Algorithmus 6.1 ergibt sich somit die Menge $LastActivities = \{d\}$. Algorithmus 6.2 entfernt dementsprechend die Einträge zu den Aktivitäten a, b, c und d aus der für diese Migration relevanten Historie (mit Einträgen zu a, b, c, d und h), so dass nur die Historieneinträge der Aktivität h übertragen werden. Da bei der Migration $M_{f,g}$ Historieneinträge zu den Aktivitäten c, d, e und f übermittelt wurden (s.o.), findet also keine redundante Informationsübertragung statt. Im Allgemeinen ist die redundante Übertragung von Historieneinträgen bei dem beschriebenen Verfahren stets ausgeschlossen: Ist ein Historieneintrag zu einer Aktivität n am Zielserver der Migration bereits bekannt, so wird dieser in der Schleife von Algorithmus 6.1 aus $OldHistory$ entfernt (sonst wäre die Schleife noch nicht verlassen worden). Deshalb wird der entsprechende Eintrag beim Durchlaufen der Schleife von Algorithmus 6.2 auch aus $MigrHistory$ entfernt und damit nicht übertragen.

6.3 Übertragung von Datenelementen

Nachdem wir in Abschnitt 6.2 untersucht haben, wie die Kontroll- bzw. Statusdaten einer WF-Instanz bei Migrationen effizient übertragen werden können, wenden wir uns nun der Übertragung von Datenelementen (vgl. Abschnitt 2.3.2) zu. Bei dem in Abschnitt 3.3.4 eingeführten Verfahren und bei den meisten der in der Literatur diskutierten Ansätze werden beim Übergang zwischen zwei Partitionen die Werte aller Datenelemente zur Nachfolgerpartition übertragen. Dies führt zu einer starken Belastung des Kommunikationssystems, insbesondere dann, wenn sehr große Datenelemente (z.B. multimediale Dokumente) verwendet und vom WfMS nicht nur Referenzen auf diese Daten verwaltet werden.[3] Im Folgenden werden Verfahren vorgestellt, mit denen die bei der Übertragung von Datenelementen entstehende Belastung des Kommunikationssystems deutlich reduziert werden kann. Bei der Entwicklung dieser Verfahren muss beachtet werden, dass von einem WfMS sowohl recht kleine als auch sehr große Datenelemente verwaltet werden können, und dass nicht jedes Verfahren für beide Fälle gleich gut geeignet ist. Zusätzliche Schwierigkeiten ergeben sich aus der Tatsache, dass die

Wird in dem Beispiel die Migration $M_{h,i}$ vor $M_{f,g}$ ausgeführt, so ergibt sich dasselbe Verhalten.

[3] Die Nachteile, die sich ergeben, wenn ein WfMS nur Referenzen auf Daten verwaltet, wurden in Abschnitt 2.3.1 schon ausführlich diskutiert.

Verfahren nicht auf den Fall statischer Serverzuordnungen beschränkt sein dürfen, sondern auch bei Verwendung variabler Serverzuordnungen anwendbar sein müssen.

6.3.1 Kleine Datenelemente

Viele der von einem WfMS verwalteten Datenelemente, wie Integer-Werte oder kurze Strings, sind relativ „klein“, so dass für sie die im nachfolgenden Abschnitt vorgestellten Optimierungen nicht lohnend sind. Hier macht es keinen Sinn, die ohnehin bereits geringe Datenmenge noch weiter zu reduzieren und dabei zu riskieren, dass zusätzliche Kommunikationszyklen notwendig werden. In diesem Abschnitt wird ein Verfahren vorgestellt, das keine zusätzlichen Kommunikationszyklen erfordert, das ohne großen Berechnungsaufwand auskommt und das vermeidet, dass ein kleines Datenelement von mehreren Quellservern an den Zielserver einer Migration übertragen wird. Die (durchschnittliche) Größe, bis zu der ein Datenelement als klein gilt, kann vom Administrator des WfMS konfiguriert werden, so dass sich ein möglichst günstiges Laufzeitverhalten ergibt. In der Regel ist es sinnvoll, ein Datenelement der Menge der großen Datenelemente $LargeDataElements$ zuzuordnen, wenn es mehrere Netzwerkpakete ausfüllt. Andernfalls wird es der in diesem Abschnitt betrachteten Menge $SmallDataElements$ zugeordnet.

Die Menge, der bei einer Migration $M_{SourceAct,TargetAct}$ zusammen mit den Ablaufhistorieneinträgen $MigrHistory$ zu übertragenden kleinen Datenelemente ($MigrDataElements$), kann mit Algorithmus 6.3 ermittelt werden. Er basiert auf der folgenden Idee: Für alle Datenelemente $d \in SmallDataElements$ kann diejenige Aktivitäteninstanz a bestimmt werden, die den für das Migrationsziel (d.h. den für die Zielaktivität $TargetAct$ der Migration) gültigen Wert von d geschrieben hat. Diese Aktivitäteninstanz kann unter Verwendung der WF-Vorlage und der Ablaufhistorie der WF-Instanz ermittelt werden. Das Datenelement d wird bei der betrachteten Migration genau dann übertragen, wenn der Historieneintrag für die Beendigung der Aktivität a in der bei der Migration übertragenen Ablaufhistorie $MigrHistory$ enthalten ist (siehe Abschnitt 6.2.2). Da sichergestellt ist, dass ein solcher Historieneintrag höchstens einmal zu jedem Server übertragen wird, gilt dies auch für jede Version eines Datenelements. Bei dem vorgestellten Verfahren ist eine redundante Übertragung von Datenelementen somit ausgeschlossen. Des Weiteren wird kein zusätzlicher Kommunikationszyklus benötigt, da die Datenelemente der Menge $MigrDataElements$ zusammen mit der Menge der Historieneinträge $MigrHistory$ zum Migrationszielserver übertragen werden. Es kann aber vorkommen, dass Datenelemente zu diesem Server übertragen werden, obwohl sie dort überhaupt nicht benötigt werden. Da die hier betrachteten Datenelemente klein sind, ist dies akzeptabel. Ein Verfahren, bei dem solche unnötigen Übertragungen ausgeschlossen sind, wird im folgenden Abschnitt für große Datenelemente vorgestellt.

Angenommen bei dem in Abb. 6.3 dargestellten Beispiel wird die Migration $M_{b,d}$ vor $M_{c,d}$ ausgeführt. Dann werden bei Verwendung des in Abschnitt 6.2.2 beschriebenen Verfahrens bei der Migration $M_{b,d}$ die Historieneinträge der Aktivitäten a und b zum Server s_2 übertragen. Damit wird das von Aktivität a geschriebene kleine Datenelement d_1 bei dieser Migration ebenfalls übertragen. Bei der Migration $M_{c,d}$ werden die Historieneinträge von Aktivität c übertragen, weshalb auch das von c geschriebene Datenelement d_2 übertragen wird. Das Datenelement d_1 wird bei dieser Migration aber nicht (redundant) an den Server s_2 übertragen.

Algorithmus 6.3 (Übertragung kleiner Datenelemente)
input
$TargetAct$: Zielaktivitäteninstanz der Migration
$SmallDataElements$: Menge der am Quellserver bekannten kleinen Datenelemente
$MigrHistory$: der zu übertragende Teil der Ablaufhistorie (von Algorithmus 6.2 ermittelt)
output
$MigrDataElements$: bei der Migration zu übertragende kleine Datenelemente
begin
$MigrDataElements = \emptyset$;
for each $d \in SmallDataElements$ **do**
// a hat diejenige Version von d geschrieben, die für $TargetAct$ gültig ist
$a = LastWriter(d, TargetAct)$;
// der zu a gehörende Ende-Historieneintrag wird bei der betrachteten Migration übertragen
if $END(a, \ldots) \in MigrHistory$ **then**
$MigrDataElements = MigrDataElements \cup \{d\}$;
end.

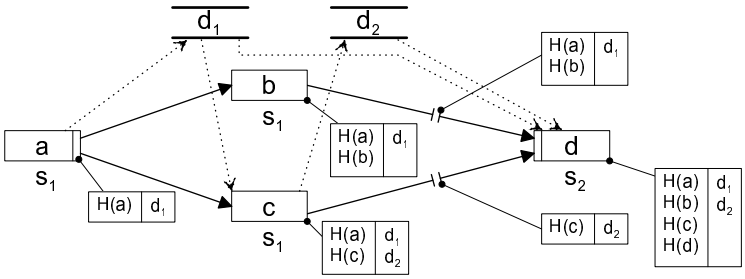


Abbildung 6.3 Migration kleiner Datenelemente, wobei die Migration $M_{b,d}$ vor $M_{c,d}$ ausgeführt wird. (Bei den Aktivitäten bzw. Migrationen ist links im Rechteck angegeben, welche Ablaufhistorieneinträge existieren bzw. übertragen werden ($H(a)$ steht für alle Ablaufhistorieneinträge der Aktivität a). Außerdem ist rechts angegeben, welche Datenelemente existieren bzw. bei einer Migration übertragen werden.)

6.3.2 Große Datenelemente

Da für die Übertragung großer Datenelemente meist mehrere Netzwerkpakete benötigt werden, ist eine zusätzliche Kommunikation akzeptabel. Es ist außerdem sehr rentabel, wenn ein solches Datenelement bei einer Migration nicht zum Zielserver transportiert werden muss, falls dieser es nicht benötigt. In dem Beispiel aus Abb. 6.4 wird das Datenelement d_1 von der Aktivität a geschrieben und von c gelesen (aber nicht von b). Deshalb können Kommunikationskosten eingespart werden, wenn das Datenelement d_1 direkt vom Server s_1 zum Server s_3 transportiert wird (ohne den Umweg über den Server s_2 der Aktivität b). Um dies zu ermöglichen, wird im Folgenden ein Verfahren entwickelt, bei dem nicht nur die redundante Übertragung von Datenelementen ausgeschlossen wird, sondern auch nur solche Datenelemente zu einem Server transportiert werden, die von diesem tatsächlich benötigt werden. Dies ist insbesondere deshalb nicht trivial, weil im Falle von bedingten Verzweigungen im Voraus nicht feststeht, welche Aktivitäten der Zielpartition einer Migration tatsächlich ausgeführt werden, und damit, welche Datenelemente benötigt werden.

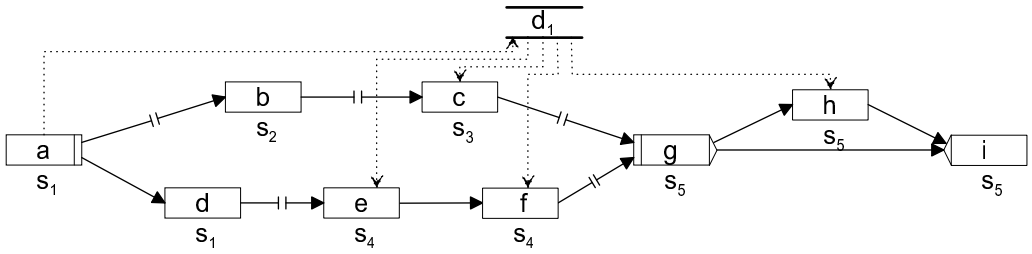


Abbildung 6.4 Übertragung von großen Datenelementen bei Migrationen.

6.3.2.1 Versenden von Datenelementen

In [Zei99] wird ein Verfahren beschrieben, bei dem – nach Ausführung der „letzten“ Aktivität einer Partition – ein in dieser Partition geschriebenes Datenelement an die WF-Server derjenigen Partitionen gesendet wird, in denen dieses Datenelement potentiell gelesen wird. In dem Beispiel aus Abb. 6.4 würde das Datenelement d_1 also (nach Beendigung der letzten Aktivität a) vom Server s_1 zum Server s_3 gesendet werden, da es in dessen Partition von der Aktivität c gelesen wird. Dieses Verfahren weist allerdings einige Nachteile auf: So lassen sich Fälle konstruieren, in denen (ebenso wie beim Versenden der Ablaufhistorie) redundante Übertragungen auftreten. Außerdem ist das Verfahren bei variablen (und bei jeder Art von dynamischen) Serverzuordnungen nicht einsetzbar. Der Grund dafür ist, dass zum Zeitpunkt des Versendens eines Datenelements, der Server der im Ablauf evtl. erst viel später folgenden Zielpartition noch gar nicht bekannt ist. Schließlich weist dieses Verfahren ein schlechteres Kommunikationsverhalten auf, als der im nachfolgenden Abschnitt beschriebene Ansatz, da ein Datenelement ausschließlich von demjenigen WF-Server bezogen werden kann, auf dem es geschrieben wurde.

6.3.2.2 Anfordern von Datenelementen

Um die gesamte bei der Aktivitätenausführung vorhandene Zustandsinformation nutzen zu können, ist es vorteilhaft, große Datenelemente bei einer Migration nicht zu versenden, sondern sie anzufordern. Um dies zu realisieren, sind zwei Verfahren denkbar: (1) Alle von einer Partition benötigten Datenelemente werden bei ihrer Aktivierung angefordert. (2) Vor dem Start jeder Aktivitäteninstanz werden die von ihr benötigten Datenelemente angefordert. Im Zusammenhang mit bedingten Verzweigungen führt das 1. Verfahren zu Problemen, da zu Beginn einer Partition evtl. noch nicht feststeht, welche Datenelemente tatsächlich benötigt werden. Im Folgenden gehen wir deshalb auf das 2. Verfahren näher ein.

Eine Aktivitäteninstanz kann in ADEPT erst dann aktiviert werden, wenn alle eingehenden Kanten signalisiert wurden. Zu diesem Zeitpunkt sind alle Vorgängeraktivitäten beendet. Deshalb kann ein Datenelement prinzipiell vom WF-Server jeder Aktivität angefordert werden, welche es geschrieben oder gelesen hat. Im Beispiel aus Abb. 6.4 kann das von der Aktivität h benötigte Datenelement d_1 vom Server der Aktivität a und von den Servern der parallel ausgeführten Aktivitäten c bzw. e und f angefordert werden. Im Allgemeinen können die WF-Server, die über eine bestimmte Version eines Datenelements verfügen, mit Hilfe der zu diesem Zeitpunkt schon übertragenen Ablaufhistorie (siehe Abschnitt 6.2.2) ermittelt werden. Das Datenelement wird dann von demjenigen WF-Server angefordert, zu dem die beste Netzwerkverbindung besteht. Um diesen Server bestimmen zu können, wird eine „Kommunikationskosten-Matrix“ vorgegeben, so dass ggf. WAN-Kommunikation vermieden werden kann.

Bevor für eine Aktivitäteninstanz Datenelemente angefordert werden, wird das in Abschnitt 6.2.2 beschriebene Verfahren zur Migration von Zustandsinformation für alle Vorgängeraktivitäten abgeschlossen (sofern diese überhaupt von einem fremden WF-Server kontrolliert wurden). Somit ist die relevante Ablaufhistorie für die zu aktivierende Aktivität $ActualAct$ bekannt. Mit dem Algorithmus 6.4 kann dann ermittelt werden, welche der von dieser Aktivität noch benötigten Datenelemente ($RequiredDataElements$) von welchem WF-Server angefordert werden sollen. Dazu wird für jedes Datenelement d dieser Menge die Aktivitäteninstanz w bestimmt, von welcher die für $ActualAct$ gültige Version des Datenelements geschrieben wurde. Die Aktivität w ist also der „letzte“ Vorgänger von $ActualAct$, der das Datenelement d geschrieben hat. Vom WF-Server dieser Aktivität könnte das Datenelement nun angefordert werden. Darüber hinaus verfügen auch die Server aller Aktivitäteninstanzen $r \in R$, welche diese Version des Datenelements gelesen haben, über dessen aktuellen Wert. Diese Aktivitäten $r \in R$ sind diejenigen, welche lesend auf d zugegriffen haben und außerdem im Kontrollfluss auf w folgen. Das Datenelement d kann also vom WF-Server der Aktivität w und von den Servern der Aktivitäten $r \in R$ angefordert werden. Von diesen Servern $s \in Sources$ wird derjenige ausgewählt, der die beste Kommunikationsverbindung zu dem Server aufweist, der $ActualAct$ kontrolliert und damit der Empfänger der Datenelemente ist.

Algorithmus 6.4 (Anfordern großer Datenelemente)
input
$ActualAct$: die aktuell auszuführende Aktivitäteninstanz
$LargeDataElements$: Menge der großen Datenelemente
$LocalDataElements$: Menge der auf dem lokalen Server schon vorhandenen Datenelemente
output
$OptServers$: Menge von Tupeln $< d, s >$, die angeben, von welchem Server s das große Datenelement d angefordert werden soll
begin
$OptServers = \emptyset$;
// Menge der von der Aktivitäteninstanz $ActualAct$ gelesenen großen Datenelemente
$RequiredDataElements = \{d \in LargeDataElements \mid d \in ReadSet(ActualAct)\}$;
// schon auf dem lokalen Server vorhandene Datenelemente nicht mehr anfordern
$RequiredDataElements = RequiredDataElements - LocalDataElements$;
for each $d \in RequiredDataElements$ **do**
// w hat diejenige Version von d geschrieben, die für $ActualAct$ gültig ist
$w = LastWriter(d, ActualAct)$;
// R enthält diejenigen Aktivitäteninst., die das von w geschriebene Datenelement d gelesen haben
$R = \{r \in SuccInst^{*}_{CONTROL\_E,SYNC\_E,LOOP\_E}(w) \mid r \in Reader(d)\}$;
// s_opt ist der bzgl. der Kommunikationskosten vom lokalen Server aus am günstigsten erreichbare
// Server, von dem d bezogen werden kann
$Sources = \{Server_w\} \cup \{Server_r \mid r \in R\}$;
wähle $s\_opt \in Sources$ mit $CommQual(Server_{ActualAct}, s\_opt)$ ist minimal;
$OptServers = OptServers \cup \{< d, s\_opt >\}$;
end.

Bei dem vorgestellten Verfahren könnten Datenelemente redundant übertragen werden, wenn durch einen WF-Server für mehrere parallele Zweige dasselbe Datenelement angefordert werden würde. Dies wird verhindert, indem Datenelemente derselben WF-Instanz nicht überlappend angefordert werden. Dann kann es zu keiner redundanten Übertragung mehr kommen, da nur Versionen von Datenelementen angefordert werden, die auf dem Zielserver noch nicht vorhanden sind. So wird in dem Beispiel aus Abb. 6.4 bei der Aktivierung der Aktivität f das Datenelement d_1 nicht angefordert, weil

es von der Aktivität e gelesen wurde und deshalb auf diesem Server schon vorhanden ist. Ein Datenelement wird erst bei der Aktivierung derjenigen Aktivität angefordert, welche es liest. Dies hat den Vorteil, dass es nur dann übertragen werden muss, wenn diese Aktivität auch tatsächlich aktiviert wird. Wird in dem Beispiel aus Abb. 6.4 der Zweig mit der Aktivität h nicht gewählt, so benötigt der Server s_5 das Datenelement d_1 nicht. Bei dem vorgestellten Verfahren wird d_1 durch den Server s_5 dann auch nicht angefordert. Der Algorithmus 6.4 wird nämlich für $ActualAct = h$ niemals ausgeführt, da die Aktivität h nicht aktiviert wird.

Dass große Datenelemente für jede Aktivitäteninstanz einzeln angefordert werden, führt zu einer Verzögerung bei der Aktivierung dieser Aktivitäteninstanz. Da die Aktivität zu diesem Zeitpunkt noch nicht in die Arbeitslisten der Benutzer eingetragen wurde, bemerken sie diese Verzögerung nicht, weshalb sie unkritisch ist. Ist ein WF-Server, von dem ein Datenelement angefordert werden soll, für längere Zeit nicht erreichbar, so kann dies die Verfügbarkeit des WfMS verschlechtern. Um dem entgegenzuwirken, sollte das Datenelement in diesem Fall (falls möglich) von einem anderen Server aus der Menge $Sources$ angefordert werden. Da die von einer Aktivität benötigen Datenelemente erst bei ihrer Aktivierung angefordert werden, entsteht ein Problem, wenn disconnected Clients [AGK+96] verwendet werden. Diese realisieren selbst einen WF-Server und sollen evtl. mehrere Aktivitäten ausführen können, ohne mit anderen WF-Servern zu kommunizieren. Deshalb müssen die Datenelemente, die von den im Disconnected-Modus auszuführenden Aktivitäten benötigt werden, schon vor dem Verbindungsabbau angefordert werden. Dabei muss in Kauf genommen werden, dass auch Datenelemente für möglicherweise nicht ausgeführte Aktivitäten (z.B. Aktivität h aus Abb. 6.4) angefordert werden müssen. Dieser Nachteil tritt aber auch bei klassischen Verfahren zur Migration von Datenelementen auf, dann aber nicht nur im Disconnected-Modus.

6.4 Zusammenfassung

Durch die bei der verteilten WF-Ausführung notwendigen Migrationen entstehen gewisse Kommunikationskosten, die reduziert werden müssen, um ein effizientes verteiltes WF-Management zu ermöglichen. In diesem Kapitel wurden Verfahren vorgestellt, mit denen das bei Migrationen zu übertragende Datenvolumen drastisch reduziert werden kann. Mit diesen Verfahren wird sowohl die redundante Übertragung interner Zustandsinformation, als auch der Parameterdaten von Aktivitätenprogrammen verhindert. Kleine Datenelemente werden anders behandelt als große, welche stets von demjenigen WF-Server bezogen werden, zu dem die beste Kommunikationsverbindung besteht. Die Verfahren sind für statische und für variable Serverzuordnungen gleichermaßen geeignet und berücksichtigen alle Konstrukte des ADEPT-Modells (z.B. bedingte und parallele Verzweigungen, Schleifen). Sie sollten deshalb auch auf andere Modelle übertragbar sein. Insbesondere im Zusammenhang mit Schleifen kann die Belastung des dem WfMS zugrunde liegenden Kommunikationssystems drastisch reduziert werden, weil mehrere Instanzen einer Aktivität vom selben Server kontrolliert werden, so dass bereits viel Information lokal vorhanden ist. Allerdings haben die vorgestellten Optimierungen auch ihren Preis: Die Anwendung der vorgestellten Verfahren erfordert zusätzlichen Rechenaufwand. Sie sollten deshalb nur in Umgebungen eingesetzt werden, in denen dieser Aufwand durch die Entlastung des Kommunikationssystems gerechtfertigt wird. Für weiträumig verteilte Anwendungen ist dies sicherlich der Fall. Die vorgestellten Verfahren reduzieren die zu übertragende Datenmenge durch die Nutzung von Wissen über schon transferierte Daten. Dadurch lässt sich eine deutlich stärkere Reduktion der Datenmenge erreichen, als mit klassischen Komprimierungsverfahren

[HM88, LH87, Sto88]. Selbstverständlich kann mit diesen die zu übertragende Datenmenge aber noch weiter reduziert werden.

Eine Diskussion der bearbeiteten Themen im Kontext allgemeiner WF-Literatur findet zentral in Kapitel 10 statt. Für das im vorliegenden Kapitel betrachtete Thema findet sie sich in Abschnitt 10.3.

Kapitel 7

Replikation von Workflow-Servern

In den Kapiteln 3 bis 6 haben wir uns darauf konzentriert, Verfahren zu entwickeln, um die in einem WfMS anfallende Kommunikationslast zu minimieren. So wurde durch die Wahl einer geeigneten Systeminfrastruktur die Grundlage dafür geschaffen, dass durch die Festlegung von entsprechenden Serverzuordnungen zur Modellierungszeit eine effiziente WF-Ausführung ermöglicht wird. Außerdem wurden Verfahren entwickelt, um die bei Migrationen entstehenden Kommunikationskosten zu reduzieren. Aufgrund dieser Maßnahmen kann sichergestellt werden, dass zur Ausführungszeit der WF-Instanzen kein Teilnetz oder Gateway überlastet wird, sofern ausreichend viele Teilnetze verwendet werden. Allerdings wurde bisher der Aspekt der Überlastung von WF-Servern vernachlässigt. Deshalb wird dieses Problemfeld im vorliegenden Kapitel ausführlich betrachtet. Dabei ist zu beachten, dass durch die Verfahren zur Vermeidung von Serverüberlastungen keine gravierenden Nachteile für das Kommunikationsverhalten des WfMS entstehen dürfen, d.h. zusätzliche Kommunikation sollte möglichst vermieden werden.

Bei der verteilten WF-Ausführung, so wie sie in dieser Arbeit vorgestellt wurde, befindet sich in jedem Domain des WfMS ein WF-Server, weshalb die Gesamtlast von mehreren WF-Servern gemeinsam bewältigt wird. Da die von jedem einzelnen Server zu bewältigende Last dadurch reduziert wird, nimmt die Wahrscheinlichkeit einer Überlastung von WF-Servern ab. Dies ist aber eher ein positiver Nebeneffekt der Verfahren zur Reduzierung der Kommunikationslast, als der Erfolg einer gezielten Entlastung der WF-Server. Durch die bisher getroffene Annahme, dass sich in jedem Domain genau ein WF-Server befindet, wurde der Fall ausgeklammert, dass für einen einzelnen Domain mehrere Server benötigt werden. Dies ist problematisch, weil ein einzelner WF-Server in einem Domain, wegen seiner begrenzten Leistungsfähigkeit, überlastet sein kann, obwohl das zugehörige Teilnetz noch nicht voll ausgelastet ist. Um dieses Problem zu lösen, könnte z.B. ein Hochleistungsrechner als WF-Server verwendet werden, oder es könnten mehrere WF-Server innerhalb eines Domains eingesetzt werden. Wie wir noch sehen werden, wird sich letzterer Ansatz als besser geeignet erweisen.

Im nächsten Abschnitt wird die in diesem Kapitel zu untersuchende Fragestellung genau festgelegt, indem die Anforderungen beschrieben werden, welche das zu entwickelnde Verfahren erfüllen muss. Außerdem wird geklärt, von welchen Annahmen und Rahmenbedingungen dabei ausgegangen werden kann. In Abschnitt 7.2 werden dann einige prinzipiell mögliche Ansätze zur Lösung des vorliegenden Problems diskutiert. Das Verfahren, mit dem für eine Aktivitäteninstanz der konkrete WF-Server eines Domains ausgewählt wird, wird in Abschnitt 7.3 vorgestellt. In Abschnitt 7.4 werden einige Fragestellungen betrachtet, welche die Veränderung einer gewählten Lastverteilung betreffen. Das Kapitel schließt mit einer Zusammenfassung und einem Ausblick auf weitere Arbeiten.

7.1 Anforderungen und Rahmenbedingungen

Um die in diesem Kapitel zu lösende Problemstellung genauer einzugrenzen, wird nun geklärt, welche Eigenschaften die zu entwickelnde Lösung aufweisen soll und von welchen Annahmen im Folgenden ausgegangen werden kann. Das Verfahren, das die Verwendung mehrerer WF-Server innerhalb eines Domains ermöglicht, soll die folgenden Anforderungen erfüllen:

1. Es muss möglich sein, die zu bewältigende Last auf beliebig viele WF-Server zu verteilen. Diese Lastaufteilung muss in einem beliebigen, vorzugebenden Verhältnis realisierbar sein. Es muss also z.B. vorgegeben werden können, dass sich die Last auf zwei zur Verfügung stehende Server im Verhältnis 3:2 aufteilt. Nur so kann bei einer beliebigen Menge an Server-Rechnern sichergestellt werden, dass keiner dieser Server überlastet wird.
2. Durch den Einsatz mehrerer WF-Server innerhalb eines Domains dürfen die durch das Verteilungsmodell von ADEPT erzielten Vorteile (z.B. die Reduzierung der Kommunikationslast) nicht konterkariert werden. Deshalb sollte das zu entwickelnde Verfahren möglichst keine (zusätzliche) Kommunikation zwischen den WF-Servern desselben Domains erfordern. Die Auswahl eines WF-Servers soll also ohne Abstimmungsprozess zwischen den verschiedenen Servern erfolgen.
3. Die Verwendung eines zentralen Servers zur Verteilung der Aufgaben verbietet sich, weil auch mit diesem kommuniziert werden müsste, und weil eine solche zentrale Komponente einen Flaschenhals darstellen und die Verfügbarkeit des WfMS beeinträchtigen würde. Es wird also ein verteiltes Verfahren zur Auswahl des Servers einer bestimmten Aktivitäteninstanz benötigt.
4. Um eine Integration in ADEPT zu ermöglichen, muss das Verfahren auch im Zusammenspiel mit variablen Serverzuordnungen funktionieren. Außerdem muss es im Falle einer dynamischen Änderung einer WF-Instanz ohne größeren Aufwand möglich sein, denjenigen Server eines Domains zu ermitteln, der die WF-Instanz tatsächlich kontrolliert (vgl. [RBD99] und Kapitel 8).
5. Es muss möglich sein, die Anzahl der WF-Server und das Verhältnis der Lastaufteilung zwischen ihnen im laufenden Betrieb zu verändern, ohne dass dieser dadurch beeinträchtigt wird.

Es ist also das Ziel, ein Verfahren zu entwickeln, das eine beliebige und veränderbare Verteilung der Last auf die WF-Server ermöglicht und dabei möglichst ohne Kommunikation auskommt. Wir erörtern nun noch einige Aspekte der im Folgenden zugrunde gelegten Systemumgebung. Da wir den operativen Einsatz von WfMS betrachten, kann von den folgenden Annahmen ausgegangen werden:

- Als WF-Server werden Rechner verwendet, die für diesen Zweck reserviert sind oder eine für diesen Zweck reservierte Bandbreite haben. Es ist also nicht anzunehmen, dass die Leistungsfähigkeit eines WF-Servers aufgrund anderer Aufgaben schwankt.
- Die Leistungsfähigkeit der WF-Server eines Domains ist etwas größer als die zu bewältigende Last erfordert (Reserve), so dass Lastschwankungen in einem gewissen Umfang ausgeglichen werden können. Dies gilt sowohl für Schwankungen der zu bewältigenden Last als auch für Schwankungen bei der Aufteilung der Last zwischen diesen Servern.
- Bei Ausfall eines WF-Servers werden dessen Aufgaben von einem Backup-Server übernommen (vgl. [KAGM96]). Die Erhöhung der Verfügbarkeit ist kein primäres Ziel der in diesem Kapitel vorgestellten Verfahren.

7.2 Lösungsansätze

Im Folgenden wird untersucht, ob der einleitend angedachte Ansatz, mehrere WF-Server innerhalb desselben Domains einzusetzen, tatsächlich am besten geeignet ist, um die vorliegende Problemstellung zu lösen. Zu diesem Zweck werden zunächst verschiedene grundlegende Ansätze vorgestellt und analysiert.

7.2.1 Verwendung eines Hochleistungssystems

Die zu bewältigende Last wird bei diesem Ansatz nicht, wie oben skizziert, auf mehrere WF-Server verteilt. Stattdessen wird ein einziger, extrem leistungsstarker Rechner verwendet.

Der Vorteil dieses Ansatzes ist, dass die bisher in dieser Arbeit entwickelten Verfahren unverändert weiterverwendet werden können und keine zusätzlichen Maßnahmen getroffen werden müssen. Allerdings entstehen für Hochleistungssysteme in der Regel sehr hohe Anschaffungskosten, so dass es sich lohnt, andere Alternativen zu betrachten. Außerdem kann auch ein solcher Rechner an seine Leistungsgrenzen stoßen.

7.2.2 Verändern der Serverzuordnungen

Bei diesem Ansatz befindet sich in jedem Domain weiterhin nur ein WF-Server. Um diesen bei Bedarf zu entlasten, können einige der von ihm kontrollierten Aktivitäten Servern anderer Domains zugeordnet werden. Dazu werden die Serverzuordnungen dieser Aktivitäten entsprechend verändert.

Durch diese Vorgehensweise werden die Kommunikationskosten erhöht, da ein Server ja gerade deshalb gewählt wurde, weil durch seine Verwendung die geringsten Kommunikationskosten anfallen. Dieses Vorgehen widerspricht damit der Kernidee von ADEPT$_{distribution}$. Außerdem ist das Verfahren überhaupt nicht anwendbar ist, wenn eine „globale Überlastung“ der WF-Server vorliegt, d.h., wenn insgesamt mehr Last zu bewältigen ist, als die aktuell vorhandenen WF-Server verarbeiten können. Da dieser Ansatz nicht vorsieht, dass mehrere WF-Server im selben Teilnetz eingesetzt werden, kann deren Gesamtkapazität nicht durch die Hinzunahme weiterer Server erhöht werden.

7.2.3 Aufspaltung eines Domains

Ist der WF-Server eines Domains überlastet, so wird dieser in mehrere Domains aufgeteilt. Die dadurch entstehenden Domains verfügen über jeweils einen WF-Server und ein eigenes Teilnetz. Bei diesem Ansatz werden also WF-Server und Teilnetze hinzugenommen. Eine Variante dieses Ansatzes ist die Bildung von logischen Domains, die zwar jeweils über einen eigenen WF-Server verfügen, denen aber dasselbe Teilnetz zugrunde liegt. Bei beiden Varianten werden die Benutzer des „Originaldomains“ auf die resultierenden Domains aufgeteilt.

Dieses Vorgehen ist immer dann empfehlenswert, wenn es eine „natürliche Zerlegung“ des Domains gibt, d.h., wenn der Domain in weitgehend disjunkte Teile (getrennte Aktivitäten und getrennte Benutzer) aufgespalten werden kann. Ist zudem auch das zugehörige Teilnetz stark belastet, so empfiehlt sich die Aufspaltung in Domains mit getrennten Teilnetzen. Dann wird allerdings durch die Veränderung der Domains ein Umbau des Kommunikationsnetzwerks notwendig.

Beide Varianten haben die folgenden gravierenden Nachteile: Häufig gibt es keine natürliche Zerlegung eines Domains, so dass keine sinnvolle Aufteilung der Benutzer auf die neu entstandenen Domains existiert. Außerdem ist es schwierig, die Last im gewünschten Verhältnis auf die Domains zu verteilen, weil die Belastung eines WF-Servers davon abhängt, wie viele und welche Bearbeiter und Aktivitäten seinem Domain zugeordnet sind. Da also nicht direkt vorgegeben werden kann, in welchem Verhältnis sich die Last auf die WF-Server verteilen soll, verletzt dies die Anforderung 1 (vgl. Abschnitt 7.1).

7.2.4 Mehrere Workflow-Server in einem Domain

Bei dem oben schon erwähnten Ansatz bleiben die Domains bezüglich ihrer Benutzer und Teilnetze unverändert. Bei Bedarf werden lediglich zusätzliche WF-Server in einem Domain verwendet. Wie der Server, der eine konkrete Aktivitäteninstanz kontrollieren soll, dann bestimmt werden kann, wird im Abschnitt 7.3 diskutiert.

Bei diesem Ansatz wird bei der Modellierung nur festgelegt, in welchem Domain der WF-Server einer Aktivität liegt. Welcher konkrete Server diese kontrollieren soll, ist ein physischer Aspekt, der bei der Modellierung nicht betrachtet wird. Der Einsatz mehrerer WF-Server je Domain ermöglicht im Prinzip eine beliebige Aufteilung der Last auf die WF-Server dieses Domains. Als positiver Nebeneffekt wird auch die Verfügbarkeit verbessert, da ein Benutzer jetzt von mehreren WF-Servern seines Domains bedient wird. Fällt einer von diesen vorübergehend aus, so kann der Benutzer zumindest mit den anderen weiterarbeiten.

Die Verwendung von mehreren WF-Servern in einem Domain kann allerdings Einfluss auf das Kommunikationsverhalten beim Aktualisieren der Arbeitslisten der Benutzer[1] haben. Ob die zu kommunizierende Datenmenge größer oder kleiner wird, hängt von der für das Aktualisieren der Arbeitslisten verwendeten Methode (vgl. Abschnitt 2.5) ab. Wird die Arbeitsliste eines Bearbeiters in festen Zeitabständen aktualisiert, so wird durch das vorgestellte Verfahren eine höhere Anzahl von Kommunikationen nötig, da diese Aktualisierungen nun von mehreren Servern pro Domain durchgeführt werden müssen. Die kommunizierte Datenmenge verändert sich insgesamt gesehen jedoch nicht, da jeder WF-Server nur den von ihm verwalteten Teil der Gesamtarbeitsliste des Benutzers übertragen muss. Wird dagegen die Arbeitsliste eines Benutzers bei jeder Änderung stets sofort aktualisiert, so ändert sich die Anzahl der Kommunikationen nicht, da die Anzahl der Änderungen gleich bleibt. Da jeweils nur noch der zu diesem WF-Server gehörende Teil der Arbeitsliste (komplett) übertragen wird, wird die insgesamt kommunizierte Datenmenge sogar kleiner. Wird jeweils nur der neu hinzugekommene Eintrag übertragen (das „Delta“), so bleibt die kommunizierte Datenmenge unverändert. Zusammenfassend lässt sich also feststellen, dass die Verwendung von mehreren WF-Servern in einem Domain Auswirkungen auf das Kommunikationsverhalten beim Aktualisieren der Arbeitslisten haben kann. Diese sind nicht besonders gravierend, da die zur Aktualisierung der Arbeitslisten übertragene Datenmenge i.d.R. klein ist, verglichen mit der sonstigen Kommunikation eines WfMS (Start von Aktivitäten, Migrationen). Insbesondere ist es von der für die Aktualisierung verwendeten Methode abhängig, ob diese Auswirkungen positiv, negativ oder neutral sind.

Da die Verfahren aus Abschnitt 7.2.1 - 7.2.3 nicht geeignet sind, um das gegebene Problem zu lösen, wird nur der zuletzt vorgestellte Ansatz weiter verfolgt. Im nun folgenden Abschnitt wird beschrie-

[1] Ein Benutzer erhält in ADEPT$_{distribution}$ seine Arbeitsliste von mehreren WF-Servern aus verschiedenen bzw. gleichen Domains. Dies ist aber für den Anwendungsentwickler transparent. Um dies zu erreichen, kapselt das API des WF-Client die verteilte Arbeitslistenverwaltung und bietet die Gesamtarbeitsliste des Benutzers an.

ben, wie bei diesem Verfahren die Auswahl eines konkreten WF-Servers für eine Aktivitäteninstanz ablaufen muss, um damit eine beliebige Verteilung der Last auf die Server erreichen zu können.

7.3 Auswahl des Workflow-Servers eines Domains

Im vorherigen Abschnitt haben wir uns dafür entschieden, den WF-Server eines Domains zu replizieren. Um diese Entscheidung umsetzen zu können, muss ein Verfahren entwickelt werden, welches für eine Aktivitäteninstanz einen Server des vorgesehenen Domains auswählt. Ein solches Verfahren muss die in Abschnitt 7.1 aufgestellten Anforderungen erfüllen, damit es nicht der Zielsetzung von ADEPT$_{distribution}$ entgegenwirkt. Insbesondere muss es eine beliebige Lastaufteilung ermöglichen und die Konzepte von ADEPT, wie z.B. variable Serverzuordnungen, unterstützen. Außerdem sollte es möglichst ohne zusätzliche Kommunikationen realisiert werden. Um die Erfüllung der Anforderung 5, welche die Änderbarkeit der Lastaufteilung im laufenden Betrieb fordert, kümmern wir uns im Abschnitt 7.4. Im Folgenden werden nun einige mögliche Vorgehensweisen untersucht und ein Verfahren entwickelt, welches die Anforderungen 1-4 erfüllt.

7.3.1 Statische Festlegung des Workflow-Servers zur Modellierungszeit

Eine einfache Möglichkeit, die Last auf mehrere Server aufzuteilen, ist, den WF-Server für eine Aktivität (so wie schon bisher den Domain) im WF-Modell explizit festzulegen (z.B. Aktivität k wird von Server s_2 des Domains 7 kontrolliert). Unterschiedliche Aktivitätentypen (aus unterschiedlichen Partitionen) können so von verschiedenen WF-Servern eines Domains kontrolliert werden.

Da konkrete Server für die Aktivitäten vorgegeben werden, ist es kaum möglich, ein gewünschtes Verhältnis der Lastaufteilung zu realisieren. Wie bei den Verfahren aus Abschnitt 7.2.3 ist damit die Anforderung 1 verletzt. Außerdem kann das Verfahren nicht in Verbindung mit variablen Serverzuordnungen eingesetzt werden, weil der entsprechende Ausdruck lediglich den Domain einer Aktivität festlegt.[2] Eine explizite Festlegung eines WF-Servers innerhalb des resultierenden Domains ist deshalb nicht möglich.

7.3.2 Lastabhängige Auswahl des Workflow-Servers

Eine Aufteilung der Last auf die WF-Server lässt sich auch erreichen, indem derjenige WF-Server ausgewählt wird, der aktuell am wenigsten belastet ist (gemessen an seiner Soll-Last). Die Identifikation des entsprechenden WF-Servers erfordert, dass Lastinformation über dessen Domain verfügbar ist. Um diese zur Verfügung zu stellen, sind prinzipiell zwei Vorgehensweisen denkbar:

1. Die Lastinformation wird (periodisch oder falls möglich Huckepack [Tan92] mit sonstigen Kommunikationen) an alle WF-Server des WfMS verteilt. Soll eine Migration durchgeführt werden, so ist die Lastinformation lokal vorhanden, so dass der Zielserver ausgewählt werden kann.

[2] Wir haben bisher im Zusammenhang mit (variablen) Serverzuordnungsausdrücken davon gesprochen, dass diese den WF-Server einer Aktivität festlegen. Dies war unter der Annahme korrekt, dass sich WF-Server und Domains wechselseitig eindeutig zugeordnet sind. Da wir im vorliegenden Kapitel aber den Fall berücksichtigen, dass mehrere WF-Server zu demselben Domain gehören können, ordnet ein solcher Ausdruck einer Aktivität eigentlich einen Domain zu. Der Server innerhalb dieses Domains, welcher die Aktivität kontrollieren soll, muss gesondert bestimmt werden.

2. Die Lastinformation wird ausschließlich einem Dispatcher im jeweiligen Domain gemeldet. Soll der WF-Server für eine Aktivitäteninstanz ausgewählt werden, so führt der Dispatcher des jeweiligen Domains diese Auswahl durch.

Die erste Methode erfordert einen höheren Aufwand bei der Verteilung der Lastinformation, weil jeder WF-Server die aktuelle Belastung jedes anderen Servers kennen muss. Bei der zweiten Methode wird der Aufwand für die Verteilung der Lastinformation reduziert, da nur der Dispatcher diese Informationen benötigt. Andererseits wird ein größerer Aufwand für die Auswahl des WF-Servers nötig, da zuerst mit dem jeweiligen Dispatcher kommuniziert werden muss. Außerdem stellt der Dispatcher eine zentrale Komponente dar, welche die Verfügbarkeit seines Domains verschlechtert, da er bei jeder Migration benötigt wird. Beide Varianten verletzen also die Anforderung 2, die zweite Methode zusätzlich die Anforderung 3.

Der Vorteil von lastabhängigen Verfahren ist, dass Schwankungen in der Belastung der WF-Server ausgeglichen werden können. Diese können aufgrund WfMS-externer Ereignisse auftreten, oder weil die von einem Server verwalteten WF-Instanzen in einem bestimmten Zeitintervall besonders viel Last erzeugen. Der entscheidende Nachteil beider Verfahren ist der zusätzlich entstehende Aufwand für Kommunikation und Synchronisation. Dieser führt dazu, dass die von den WF-Servern bewältigte Last nicht mehr linear zu deren Leistungsfähigkeit wächst. Außerdem ist – wie schon von Scheduling-Verfahren bekannt ist [Gos91] – für den Erfolg eines lastabhängigen Verfahrens die Qualität der Lastinformation und eine geeignet gewählte Frequenz für den Lastinformationsaustausch äußerst kritisch. Dies gilt umso mehr, weil WF eine komplexe interne Struktur aufweisen und für mehrere Wochen am gewählten Server verweilen können. Zusammenfassend lässt sich also feststellen, dass lastabhängige Verfahren ein (akademisch) interessanter Ansatz sind, da sie potentiell die Möglichkeit bieten, Lastschwankungen auszugleichen. Sie sind aber schwer zu realisieren und erzeugen selbst zusätzliche Kommunikationslast.

7.3.3 Zufällige Auswahl des Workflow-Servers zur Laufzeit

Wenn für eine WF-Instanz mit der Ausführung einer Partition (in dem vorgegebenen Domain) begonnen wird (WF-Start oder Migration), dann kann zufällig ein WF-Server dieses Domains ausgewählt werden. Dazu wird eine Zufallszahl z aus dem Intervall $[0, 1)$ berechnet. Dieses Intervall wird in disjunkte Teilintervalle zerlegt, die den WF-Servern des Domains zugeordnet sind. Für die fragliche Partition wird derjenige WF-Server gewählt, in dessen Teilintervall z fällt. Die Wahrscheinlichkeit, dass ein bestimmter WF-Server gewählt wird (und damit sein Anteil an der Gesamtlast), entspricht der Länge seines Intervalls.

Ein solches Verfahren hat einige sehr schöne Eigenschaften: Es ermöglicht eine beliebige Aufteilung der Last. Diese ist zudem, durch eine Veränderung der Teilintervalle, leicht änderbar. Das Verfahren ist auch für variable Serverzuordnungen anwendbar und es ist sogar möglich, die Last für die Steuerung eines einzigen Aktivitätentyps auf mehrere WF-Server zu verteilen. Wegen der großen Anzahl von WF-Instanzen ist es sehr unwahrscheinlich, dass, aufgrund von Unzulänglichkeiten des Zufallszahlengenerators, deutlich von der geplanten Lastverteilung abgewichen wird.

Das Verfahren hat allerdings einen entscheidenden Nachteil: Es führt zu einem Problem bei der Zusammenführung paralleler WF-Zweige und bei Aktivitäten mit eingehenden Synchronisationskanten. Dies soll an dem in Abb. 7.1 dargestellten Beispiel erläutert werden: Angenommen, für die aktuelle WF-Instanz findet die Migration $M_{b,c}$ vor $M_{d,e}$ statt. Dann wird bei der Ausführung von $M_{b,c}$ der

konkrete WF-Server innerhalb des Domains 3 festgelegt. Dieser ist aber dem WF-Server des Domains 2 nicht ohne weiteres bekannt. Bei einer naiven Implementierung dieses Verfahrens könnte es deshalb vorkommen, dass dieser bei der Migration $M_{d,e}$ einen anderen WF-Server aus dem Domain 3 (zufällig) auswählt, was zu einem Problem bei der Zusammenführung der parallelen Zweige an der Aktivität f führen würde. Eine Abstimmung des Zielservers zwischen den verschiedenen Startservern der Migration erfordert jedoch zusätzliche Kommunikation, was der Anforderung 2 widerspricht.

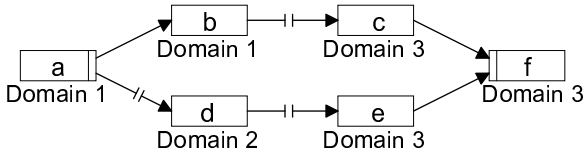


Abbildung 7.1 Problem bei der Zusammenführung paralleler Zweige bei zufälliger Serverwahl.

7.3.4 Pseudo-zufällige Auswahl des Workflow-Servers zur Laufzeit

Wir wollen die positiven Eigenschaften des vorherigen Verfahrens erhalten, jedoch das Problem bei der Zusammenführung paralleler Zweige vermeiden. Dies gelingt durch folgenden Trick: Der WF-Server wird nicht rein zufällig gewählt, sondern so, dass für eine WF-Instanz in einem Domain stets derselbe WF-Server ermittelt wird. Dazu wird ein WF-instanzspezifisches Datum (z.B. die WF-ID) mit einem Pseudo-Zufallszahlengenerator auf ein $z \in [0, 1)$ abgebildet. Dadurch ergibt sich für eine WF-Instanz stets dasselbe z und damit derselbe WF-Server.

Alle Vorteile des vorherigen Verfahren bleiben erhalten: Es wird eine beliebige Aufteilung der Last auf die WF-Server ermöglicht, alle Konzepte von ADEPT (inkl. variabler Serverzuordnungen) werden unterstützt und das Verfahren kommt ohne zusätzliche Kommunikation aus. Die Probleme bei der Zusammenführung paralleler Zweige werden vermieden, da innerhalb eines Domains für alle parallelen Zweige einer WF-Instanz derselbe Server ermittelt wird. Weil es ebenfalls keine Kommunikation erfordert, erfüllt das Verfahren die Anforderungen 1-4 aus Abschnitt 7.1. Für die Praxis ist zu erwarten, dass es besser funktioniert als lastabhängige Verfahren. Die Gründe dafür sind, dass keine zusätzliche Kommunikation notwendig wird, und dass das zu erwartende Systemverhalten leichter abschätzbar ist. Da die WF-Server über eine Lastreserve verfügen, spielt eine leichte Ungleichverteilung der Last keine große Rolle. Auch deshalb wird kein Verfahren benötigt, das eine solche Ungleichverteilung aktiv ausgleicht. Aus diesen Gründen wird das pseudo-zufällige Verfahren weiter verfolgt. Im Folgenden wird noch geklärt, wie die geforderte pseudo-zufällige Aufteilungsfunktion realisiert und die von den WF-Servern benötigte Information bereitgestellt werden kann. In Abschnitt 7.4 wird dann diskutiert, wie zusätzlich die Anforderung 5 (Änderbarkeit der Lastaufteilung) erfüllt werden kann.

7.3.4.1 Realisierung der Aufteilungsfunktion

Bei dem Verfahren wird ein instanzspezifisches Datum der WF-Instanz *Inst* verwendet, um z.B. bei einer Migration einen WF-Server des Zieldomains D auszuwählen. Diese Berechnung übernimmt die domainspezifische Aufteilungsfunktion $g^D(\mathit{Inst})$ (siehe Algorithmus 7.1). Das unveränderliche instanzspezifische Datum *InstDat* (z.B. die WF-ID) der WF-Instanz *Inst* wird mit Hilfe eines Pseudo-Zufallszahlengerators auf einen Wert z im Intervall $[0, 1)$ abgebildet. Die Funktion $f^D(z)$ bildet diesen Wert z auf einen Server s_i des Domains D ab. Durch die Wahl dieser Funktion f lässt sich die

Lastverteilung auf die WF-Server steuern. Da ein Pseudo-Zufallszahlengerator ausgehend vom selben Startwert *InstDat* stets dasselbe Ergebnis z berechnet, ergibt sich für eine WF-Instanz *Inst* stets derselbe WF-Server s_i im Domain D.

Algorithmus 7.1 (Berechnung der Aufteilungsfunktion $g^D(Inst)$)
input
Inst: WF-Instanz, für die der WF-Server ermittelt werden soll
D: Domain, aus dem ein WF-Server gewählt werden soll
result
logische Server-ID des für die WF-Instanz *Inst* ausgewählten WF-Servers aus dem Domain D
begin
InstDat $= instanzspezifisches\_Datum($*Inst*$)$;
$z = pseudo\_random($*InstDat*$)$;
Funktion $f^D(z)$ definiert durch $a_1^D \ldots a_{n-1}^D$ und $s_1 \ldots s_n$:
case $z \in [0, a_1^D)$: **return** s_1;
case $z \in [a_1^D, a_2^D)$: **return** s_2;
...
case $z \in [a_{n-1}^D, 1)$: **return** s_n;
end.

Die Wahrscheinlichkeit für die Auswahl des WF-Servers s_i durch den Algorithmus 7.1 entspricht der Länge des zugehörigen Intervalls $[a_{i-1}, a_i)$. Dieser Aussage liegt die Annahme zugrunde, dass die Werte von z über $[0, 1)$ gleichverteilt sind. Dies kann durch den Einsatz eines „guten“ Pseudo-Zufallszahlengenerators bei der Berechnung der Funktion $pseudo\_random$ erreicht werden. In Algorithmus 7.1 kann als instanzspezifisches Datum die z.B. die ID der WF-Instanz verwendet werden. Stattdessen kann auch die Startzeit der WF-Instanz (oder ein speziell für diesen Zweck generierter Zufallswert) gewählt werden. Wird z.B. von der Startzeit nur der Wert für die Millisekunden als *InstDat* verwendet, so ist es realistisch anzunehmen, dass die Werte von *InstDat* gleichmäßig über das Intervall $[0, 1000)$ verteilt sind. Deshalb ist in diesem Fall die Funktion $pseudo\_random($*InstDat*$) :=$ *InstDat*$/1000$ ausreichend, um eine Gleichverteilung von z über $[0, 1)$ zu erreichen.

7.3.4.2 Verteilung der benötigten Information

Beim Start von WF-Instanzen und bei deren Migration muss entschieden werden können, welcher Server s des Domains D für die WF-Instanz *Inst* verwendet werden soll. Dazu muss die Funktion $g^D($*Inst*$)$ allen WF-Servern des WfMS bekannt sein. Diese Funktion berechnet eine logische Server-ID, die mit einer anderen Funktion $h(s)$ auf eine physische Rechneradresse (z.B. IP-Adresse) abgebildet wird. Auch $h(s)$ muss allen Servern bekannt sein. Da diese beiden Funktionen selten geändert werden, werden sie auf allen WF-Servern repliziert (ähnlich wie die WF-Vorlagen). Aspekte, welche die Änderung dieser Funktionen betreffen, werden im nächsten Abschnitt diskutiert. Da solche Änderungen eher selten auftreten, fallen die durch die Replikation entstehenden Kommunikationskosten nicht ins Gewicht.

7.4 Änderung der Lastaufteilung

Das in Abschnitt 7.3.4 vorgestellte, pseudo-zufällige Verfahren erfüllt die Anforderungen 1-4. In der oben beschriebenen Form wird die Anforderung 5, dass die Aufteilung der Last veränderbar sein

muss, noch nicht erfüllt. Dies ist aber notwendig, damit eine existierende Überlastung behoben oder eine Systemkonfiguration verändert werden kann. Die Aufteilungsfunktion zu verändern ist allerdings nicht unproblematisch, weil eine WF-Instanz stets vom selben Server innerhalb eines Domain kontrolliert werden muss, um das in Abschnitt 7.3.3 beschriebene Problem beim Zusammenführen paralleler Zweige zu vermeiden. Ungünstigerweise erfordert eine Veränderung der Lastaufteilung jedoch eine Veränderung dieser Zuordnung. Im Folgenden wird untersucht, wie Server ausgetauscht, hinzugefügt und weggenommen werden können und wie die Lastaufteilung verändert werden kann, so dass die Anforderung 5 erfüllt wird, ohne dadurch die Anforderungen 1-4 zu verletzen.

7.4.1 Physische Server austauschen

Soll der WF-Server mit der logischen ID s_{change} durch einen anderen ersetzt werden, so wird dieser durch Algorithmus 7.2 zuerst gestartet. Nachdem die Möglichkeit zur Migration zu dem von der Änderung betroffenen Server gesperrt wurde, wird die Abbildungsfunktion $h(s)$ der logischen Server-IDs auf die Rechneradressen verändert. Dann werden die WF-Instanzen, welche vom stillzulegenden Server kontrolliert werden, zum neuen Server transportiert. Die neue Funktion $h_{(n+1)}(s)$, welche die bisher gültige Funktion $h_{(n)}(s)$ ersetzt, wird repliziert, so dass sie jedem Server des WfMS bekannt ist. Die Änderung der Funktion $h(s)$ bewirkt, dass am alten WF-Server zukünftig keine Migrationen mehr eingehen. Jetzt können die Sperren aufgehoben werden, so dass auch wieder Migrationen zum Server s_{change} stattfinden können. Schließlich kann der zu ersetzende Server gestoppt werden.

Algorithmus 7.2 (Austauschen eines WF-Servers)
input
s_{change}: logische ID des Servers, welcher ersetzt werden soll
adr: physische Adresse des Servers, welcher die logische ID s_{change} erhalten soll
$h_{(n)}(s)$: bisherige Abbildungsfunktion logischer IDs von WF-Servern auf deren physische Adresse
begin
starte den neuen WF-Server adr;
sperre (bei allen WF-Servern) die Migrationen zum (bisherigen) Server s_{change};
// ordne s_{change} die neue Server-ID zu und übernehme die anderen Einträge
$h_{(n+1)}(s_{change}) := adr$;
$\forall s \neq s_{change}$: $h_{(n+1)}(s) := h_{(n)}(s)$;
transferiere alle WF-Instanzen vom Server $h_{(n)}(s_{change})$ zum Server $h_{(n+1)}(s_{change})$;
repliziere $h_{(n+1)}(s)$ an alle WF-Server;
gebe Migrationen zum WF-Server s_{change} wieder frei;
stoppe den zu deaktivierenden WF-Server $h_{(n)}(s_{change})$;
end.

Eigentlich müsste das Sperren, das Replizieren von $h_{(n+1)}(s)$ und das Freigeben der Sperren in einer verteilten Transaktion stattfinden. Dadurch wären aber Migrationen zum Server s_{change} nicht möglich, solange die Transaktion nicht bei allen Servern beendet wurde. Der Server s_{change} würde also durch den Ausfall eines beteiligten Servers während der Ausführung von Algorithmus 7.2 blockiert werden. Außerdem würde ein solches Vorgehen die Ausführung eines teuren 2PC erforderlich machen. Um diese Nachteile zu vermeiden, wird für den Algorithmus 7.2 und die nachfolgend beschriebenen Algorithmen der folgende Trick verwendet: Die Möglichkeit zur Migration wird weiterhin bei allen WF-Servern gesperrt. Nach Ausführung des entsprechenden „Sperrbefehls“ können bei s_{change} also keine neuen Migrationen mehr eingehen. Die Replikation der Funktion $h_{(n+1)}(s)$ wird nun mit dem Freigeben der Sperre zu einer einzigen Transaktion zusammengefasst. Sobald nun ein Server

die neuen Daten erfolgreich (lokal) gespeichert hat, hebt er die Sperre für Migrationen zu s_{change} auf, so dass Migrationen zu s_{change} (auf Basis der neuen Funktion) von ihm durchgeführt werden können. Durch diese Vorgehensweise wird zwar die Sperre nicht bei allen Servern gleich lange gehalten, die Möglichkeit zur Migration ist aber entweder noch gesperrt oder es wird schon die neue Funktion $h_{(n+1)}(s)$ verwendet. Deshalb können keine Inkonsistenzen durch die Verwendung der unterschiedlichen Funktionen $h_{(n)}(s)$ und $h_{(n+1)}(s)$ entstehen. Der große Vorteil dieser Vorgehensweise ist, dass jeder WF-Server sofort nach der erfolgreichen (lokalen) Speicherung der Daten wieder ohne Einschränkung weiterarbeiten kann, auch wenn noch nicht alle Server die Änderung der Abbildungsfunktion (z.B. wegen eines lokalen Systemausfalls) vollzogen haben.

7.4.2 Lastverteilung ändern

Wenn zwar weiterhin dieselben WF-Server verwendet werden sollen, aber die Verteilung der Last zwischen ihnen geändert werden soll, so muss die bisher gültige Aufteilungsfunktion $g^D_{(n)}(\mathit{Inst})$ durch eine neue Aufteilungsfunktion $g^D_{(n+1)}(\mathit{Inst})$ ersetzt werden. Durch diese Veränderung können Probleme bei der Zusammenführung paralleler Zweige einer WF-Instanz entstehen: Wenn einzelne Zweige einer WF-Instanz die alte Funktion verwenden und andere die neue, so kann es vorkommen, dass eine WF-Instanz von verschiedenen WF-Servern eines Domains kontrolliert wird. Dies wollen wir jedoch ausschließen, um Kommunikation zwischen WF-Servern desselben Domains zu vermeiden (Anforderung 2). Zu diesem Zweck kann eines der folgenden Verfahren verwendet werden:

1. In Algorithmus 7.3 wird die neue Aufteilungsfunktion $g^D_{(n+1)}(\mathit{Inst})$ des Domains D mit einem Zeitstempel $T^D_{(n+1)}$ versehen, der mit ihr repliziert wird. Die neue Funktion $g^D_{(n+1)}(\mathit{Inst})$ wird dann nur für WF-Instanzen *Inst* verwendet, für die gilt: $Startzeitpunkt(\mathit{Inst}) > T^D_{(n+1)}$. Während der Algorithmus 7.3 ausgeführt wird, ist die Möglichkeit zur Migration in den Domain D gesperrt. Für WF-Instanzen, die nach dem Start von Algorithmus 7.3 erzeugt wurden, ist damit zum Zeitpunkt einer etwaigen Migration in den Domain D die neue Funktion $g^D_{(n+1)}(\mathit{Inst})$ und der zugehörige Zeitstempel bekannt, so dass die Migration auch wirklich auf Basis der neuen Funktion $g^D_{(n+1)}(\mathit{Inst})$ durchgeführt werden kann.

Algorithmus 7.3 (1. Verfahren zur Änderung von $g^D(\mathit{Inst})$)
input
D: Domain, in dem die Lastaufteilung verändert werden soll
$g^D_{(n+1)}(\mathit{Inst})$: neue Aufteilungsfunktion für den Domains D
begin
sperre (bei allen WF-Servern) die Migrationen in den Domain D;
$T^D_{(n+1)} = time()$; // Zeitstempel auf die aktuelle Zeit setzen
repliziere $(g^D_{(n+1)}(\mathit{Inst}), T^D_{(n+1)})$;
gebe Migrationen in den Domain D wieder frei;
end.

2. Bei dem in Algorithmus 7.4 dargestellten Verfahren wird die Menge $A^D_{(n+1)}$ der IDs von WF-Instanzen berechnet, die zur Zeit der Änderung in der Datenbank eines WF-Servers s des Domains D gespeichert sind und für die sich durch die Änderung der WF-Server ändert. Für eine WF-Instanz *Inst* mit $\mathit{ID}(\mathit{Inst}) \in A^D_{(n+1)}$ wird nicht die neue Funktion $g^D_{(n+1)}(\mathit{Inst})$ verwendet, sondern weiterhin die alte Funktion $g^D_{(n)}(\mathit{Inst})$ (bzw. $g^D_{(n-1)}(\mathit{Inst})$, falls zusätzlich $\mathit{ID}(\mathit{Inst}) \in A^D_{(n)}$ gilt, usw.). Die Menge $A^D_{(n+1)}$ enthält also die IDs derjenigen WF-Instanzen, die zum Zeitpunkt

der Änderung von einem WF-Server des Domains D kontrolliert werden und für die sich durch $g^D_{(n+1)}$ der Server ändern würde.[3] Für diese WF-Instanzen wird weiterhin die „alte“ Aufteilungsfunktion verwendet, so dass der entsprechende WF-Server alle parallelen Zweige kontrolliert. Deshalb entsteht für diese Instanzen kein Problem beim Zusammenführen paralleler Zweige. Für alle anderen WF-Instanzen ($ID(Inst) \notin A^D_{(n+1)}$) wird ausschließlich die „neue“ Funktion $g^D_{(n+1)}$ verwendet, so dass auch hier kein Problem auftreten kann.

Algorithmus 7.4 (2. Verfahren zur Änderung von $g^D(Inst)$)
input
D: Domain, in dem die Lastaufteilung verändert werden soll
$g^D_{(n+1)}(Inst)$: neue Aufteilungsfunktion des Domains D
S_D: Menge der WF-Server des Domains D
begin
sperre (bei allen WF-Servern) die Migrationen in den Domain D;
$activeWFs = \bigcup_{s \in S_D} \{Inst \mid$ WF-Instanz *Inst* ist am WF-Server s aktiv$\}$;
// im Domain D aktive WF-Instanzen, für die sich der Server verändern würde:
$A^D_{(n+1)} = \{ID(Inst) \mid Inst \in activeWFs \wedge Server^D(Inst) \neq g^D_{(n+1)}(Inst)\}$;
repliziere $(g^D_{(n+1)}(Inst), A^D_{(n+1)})$;
gebe Migrationen in den Domains D wieder frei;
end.

Der Vorteil des ersten Verfahrens ist, dass es einen sehr geringen Aufwand erfordert. Allerdings wird die neue Funktion $g^D_{(n+1)}(Inst)$ erst dann für alle WF-Instanzen verwendet, wenn keine WF-Instanzen mehr existieren, die älter als diese Funktion sind. Da WF-Instanzen häufig mehrere Wochen laufen, kann es entsprechend lange dauern, bis die neu eingestellte Lastverteilung tatsächlich erreicht wird. Das zweite Verfahren erfordert wegen der Berechnung und Replikation von $A^D_{(n+1)}$ einen wesentlich größeren Aufwand als das erste Verfahren. Es hat aber den Vorteil, dass auch für schon lange laufende WF-Instanzen der Migrationszielserver auf Basis der neuen Aufteilungsfunktion ermittelt wird. Welches der beiden Verfahren besser geeignet ist, ist von der jeweiligen Situation abhängig. Soll die Lastaufteilung lediglich längerfristig angepasst werden, so ist die 1. Variante ausreichend. Sollen aber die Auswirkungen der Änderung möglichst ab sofort beobachtet und bewertet werden, so ist die lange Verzögerung nicht akzeptabel. Dann muss die 2. Variante verwendet werden. Die dabei entstehenden Kosten relativieren sich, da Veränderungen der Lastaufteilung i.d.R. doch eher selten sind.

7.4.3 Anzahl der Server eines Domains erhöhen

Soll die Anzahl der WF-Server eines Domains D erhöht werden, so muss eine neue logische Server-ID s_{new} definiert werden. Nachdem der Algorithmus 7.5 den neuen WF-Server gestartet hat, wird dessen logische ID s_{new} auf dessen physische Adresse adr_{new} abgebildet. Dies geschieht durch die Änderung und Replikation der Funktion $h(s)$. Für diesen Vorgang sind keine Sperren erforderlich, weil s_{new} noch kein Intervall der Aufteilungsfunktion $g^D(Inst)$ zugeordnet ist; der Server s_{new} wird also überhaupt noch nicht verwendet. Damit er jedoch in Zukunft WF-Instanzen kontrollieren kann,

[3] Die Funktion $Server^D(Inst)$ ermittelt denjenigen WF-Server des Domains D, der die WF-Instanz *Inst* gerade kontrolliert. Dieser kann nicht direkt mit $g^D_{(n)}(Inst)$ berechnet werden, da $g^D_{(n)}$ nicht unbedingt die für *Inst* gültige Aufteilungsfunktion sein muss (dies kann z.B. auch $g^D_{(n-1)}$ sein). Zur Berechnung von $Server^D(Inst)$ muss also $A^D_{(n)}$ (und ggf. auch $A^D_{(n-1)}$, usw.) berücksichtigt werden, um die korrekte Version von $g^D(Inst)$ auswählen zu können.

muss zusätzlich die Funktion $g^D(\mathit{Inst})$, wie im vorherigen Abschnitt beschrieben, geändert werden. Dem neuen WF-Server muss also ein Intervall der Aufteilungsfunktion zugeordnet werden.

Algorithmus 7.5 (Neuen WF-Server einführen)
input
s_{new}: logische ID des neuen WF-Servers
adr_{new}: physische Adresse des neuen Servers
$h_{(n)}(s)$: aktuelle Funktion, die logische Server-IDs auf physische Server-Adressen abbildet
$g^D_{(n'+1)}(\mathit{Inst})$: neue Aufteilungsfunktion des Domains D, in der auch s_{new} berücksichtigt wird
begin
starte den WF-Server adr_{new};
$h_{(n+1)}(s_{new}) := adr_{new}$;
$\forall s \neq s_{new}$: $h_{(n+1)}(s) := h_{(n)}(s)$;
repliziere $h_{(n+1)}(s)$;
ändere $g^D_{(n')}(\mathit{Inst})$ in $g^D_{(n'+1)}(\mathit{Inst})$ mit Algorithmus 7.3 oder 7.4;
end.

7.4.4 Anzahl der Server eines Domains reduzieren

Wenn die Anzahl der WF-Server des Domains D verringert werden soll, so verhindert der Algorithmus 7.6 zuerst durch eine Sperre weitere Migrationen zu dem stillzulegenden Server s_{old}. Dann werden alle bei ihm vorhandenen WF-Instanzen zu dem Server s_{new} transferiert, welcher dessen Aufgaben (vorläufig) übernimmt. Damit zukünftig keine Migrationen an dem stillzulegenden Server mehr eingehen, wird die logische Server-ID s_{old} auf die physische Adresse des übernehmenden WF-Servers „umgelenkt“, indem die Funktion $h(s)$ verändert wird. Nun wird die neue Funktion $h_{(n+1)}(s)$ repliziert und die Sperren werden freigegeben. Da der stillzulegende Server keine WF-Instanzen mehr kontrolliert und auch keine Migrationen mehr bei ihm eingehen können, wird er gestoppt. Um auch die logische ID des stillgelegten Servers auslaufen zu lassen, sollte nach Beendigung von Algorithmus 7.6 mit den Verfahren aus Abschnitt 7.4.2 die Aufteilungsfunktion $g^D(\mathit{Inst})$ so verändert werden, dass s_{old} kein Intervall mehr zugeordnet ist. Im Zuge dieser Veränderung kann zusätzlich die bisher von s_{old} bewältigte Last geeignet auf andere WF-Server verteilt werden, indem die Intervalle entsprechend angepasst werden.

Algorithmus 7.6 (WF-Server stilllegen)
input
s_{old}: logische ID des stillzulegenden WF-Servers
s_{new}: logische ID des Servers, der die Aufgaben von s_{old} übernehmen soll
$h_{(n)}(s)$: aktuelle Funktion, die logische Server-IDs auf physische Server-Adressen abbildet
begin
sperre (bei allen WF-Servern) die Migrationen zum Server s_{old};
$h_{(n+1)}(s_{old}) := h_{(n)}(s_{new})$; $\forall s \neq s_{old}$: $h_{(n+1)}(s) := h_{(n)}(s)$;
transferiere alle WF-Instanzen vom Server $h_{(n)}(s_{old})$ zum Server $h_{(n)}(s_{new})$;
repliziere $h_{(n+1)}(s)$;
gebe Migrationen zum WF-Server s_{old} wieder frei;
stoppe den WF-Server $h_{(n)}(s_{old})$;
end.

Wie wir gesehen haben, ist es also nicht nur möglich, die Aufteilungsfunktion der WF-Instanzen auf die WF-Server effizient und mit guten statistischen Eigenschaften zu realisieren, sie kann außerdem

im laufenden Betrieb beliebig verändert werden. Damit erfüllt das oben beschriebene Verfahren nicht nur – wie schon in Abschnitt 7.3 festgestellt wurde – die Anforderungen 1-4, sondern zusätzlich auch die Anforderung 5 aus Abschnitt 7.1.

7.5 Zusammenfassung und Ausblick

Die zahlreichen Aufgaben eines WF-Servers (vgl. Abschnitt 3.1.1) können zu dessen Überlastung führen. Um dies zu verhindern, kann ein WF-Server repliziert und die Last zwischen den entstehenden Replikaten aufgeteilt werden. In diesem Kapitel wurde ein Verfahren vorgestellt, bei dem für eine WF-Instanz pseudo-zufällig ein WF-Server ausgewählt wird. Dieses Verfahren realisiert eine Art „Hashing“ der WF-Instanzen auf die WF-Server eines Domains. Es ermöglicht eine beliebige Aufteilung der Last zwischen den WF-Servern, generiert im wesentlichen keine zusätzliche Kommunikation und erfordert nur sehr wenig Rechenaufwand. Weiter wurde gezeigt, wie es möglich ist, die Lastaufteilung zwischen den WF-Servern im laufenden Betrieb zu verändern. Damit ist das Problem der Überlastung der WF-Server gelöst, was zeigt, dass die bisher in dieser Arbeit erfolgte Konzentration auf den Kommunikationsaspekt legitim war. Da bei der Modellierung nur Domains – und nicht konkrete Rechner – vorgegeben werden, können die verschiedenen WF-Server eines Domains als ein einziger leistungsstarker Server („virtual powerful server“) betrachtet werden. Deshalb müssen die einzelnen Server eines Domains im Folgenden nicht gesondert betrachtet werden und der Begriff der Serverzuordnung macht weiterhin Sinn, obwohl durch diese einer Aktivität eigentlich ein Domain zugeordnet wird, während der WF-Server durch die hier vorgestellten Mechanismen ausgewählt wird.

In diesem Kapitel wurde auch die Anwendbarkeit lastabhängiger Verfahren zur Serverauswahl untersucht. Diese haben zwar im Allgemeinen Nachteile gegenüber dem pseudo-zufälligen Verfahren, sind aber evtl. für bestimmte Spezialanwendungen interessant. Eine vertiefte Untersuchung dieser Verfahren bietet sich vor allem an, wenn die verfügbare Leistung der WF-Server stark schwankt, wenn der Ausfall von WF-Servern durch dieses Verfahren kompensiert werden soll, oder wenn ein WF-Server nur sehr wenige WF-Instanzen gleichzeitig kontrollieren kann, so dass schon kleine Ungleichverteilungen, wie sie beim pseudo-zufällige Verfahren entstehen können, zu Problemen führen würden. Für große WF-Anwendungen, die operativ eingesetzt werden, ist aber das vorgestellte Verfahren den lastabhängigen Verfahren vorzuziehen.

Kapitel 8

Dynamische Modifikation von Workflow-Instanzen

Um WfMS für ein breites Spektrum von Anwendungen einsetzen zu können, ist es notwendig, dass zur Ausführungszeit einer WF-Instanz dynamisch von ihrem vormodellierten Ablauf abgewichen werden kann. Dies ist insbesondere in unternehmensweiten WfMS essentiell, da es in fast jedem Unternehmen gewisse WF-Typen gibt, deren Instanzen nicht ausschließlich starr entsprechend der WF-Vorlage ausgeführt werden können. Wenn nun für eine bestimmte WF-Instanz eine Ausnahmesituation eintritt, so muss es möglich sein, ihre Vorlage so umzustrukturieren, dass sie den neu entstandenen Anforderungen entspricht. Dies kann z.B. dadurch geschehen, dass zusätzliche Aktivitäten eingefügt, existierende Aktivitäten gelöscht oder die Reihenfolgebeziehungen zwischen Aktivitäten geändert werden.

Die Möglichkeit zur strukturellen Änderung von WF-Instanzen zu ihrer Ausführungszeit ist eine der wesentlichen Eigenschaften von ADEPT. Das dafür notwendige Grundgerüst bietet das ADEPT$_{flex}$-Kalkül [RD97, RD98, RHD98, DRK00, Rei00]. Allerdings setzen die entsprechenden Verfahren eine logisch zentrale Kontrollinstanz voraus. Diese wird benötigt, um überprüfen zu können, ob eine geplante Änderung überhaupt durchgeführt werden kann, und um die Konsistenz des erzeugten WF-Graphen garantieren zu können. Eine zentrale Kontrollinstanz widerspricht jedoch den Zielen von ADEPT$_{distribution}$. Deshalb wird in diesem Kapitel untersucht, wie adaptives WF-Management realisiert werden muss, um die Skalierbarkeit des WfMS nicht zu beeinträchtigen. Um weiterhin alle in der Realität auftretenden Anforderungen erfüllen zu können, soll das zu entwickelnde Verfahren die volle Funktionalität realisieren, die für dynamische Änderungen im zentralen Fall zur Verfügung steht. Eine noch ausführlichere und formalere Beschreibung der im Folgenden vorgestellten Verfahren findet sich in [BRD01a].

Im nun folgenden Abschnitt wird kurz erläutert, wie dynamische Änderungen von WF-Instanzen in ADEPT$_{flex}$ ablaufen. Abschnitt 8.2 analysiert die durch die Kombination von dynamischen Änderungen und verteilter WF-Ausführung entstehenden Probleme. In Abschnitt 8.3 wird dann aufgezeigt, wie dynamische WF-Modifikationen in einer verteilten Systemumgebung realisiert werden sollten. Das Kapitel schließt mit einer Zusammenfassung und einem Ausblick auf einige weitere Fragestellungen, die im betrachteten Kontext relevant sind.

8.1 Einführung in dynamische Änderungen

Der Grund für die Notwendigkeit einer dynamischen Änderung einer WF-Instanz ist i.d.R. das Eintreten einer Ausnahmesituation. So können z.B. bei einem klinischen WF aufgrund von Besonderheiten eines Patienten zusätzliche Untersuchungen notwendig werden oder in Notfällen Terminvereinbarungen entfallen. Im Folgenden wird erläutert, wie solche dynamischen Modifikationen in ADEPT realisiert werden.

ADEPT$_{flex}$ bietet eine vollständige und minimale Menge von Basisoperationen an (z.B. Kante einfügen, Aktivität löschen), mit denen prinzipiell jede denkbare Änderung der Struktur einer WF-Instanz zu ihrer Ausführungszeit durchgeführt werden kann. Eine solche Änderung macht allerdings nur dann Sinn, wenn die Konsistenz des entstehenden WF-Ausführungsgraphen gewährleistet werden kann. So darf in dem WF-Graphen z.B. kein Zyklus über Kontroll- und Synchronisationskanten entstehen, weil dies zu einer Verklemmung führen würde, und der Datenfluss muss korrekt sein, weil bei nicht versorgten Eingabeparametern die zugehörigen Aktivitätenprogramme fehlerhaft arbeiten oder sogar abstürzen könnten. Deshalb werden in ADEPT$_{flex}$ die Voraussetzungen für die Anwendbarkeit jeder Änderungsoperation, bzgl. der Struktur und des Zustands des Ausführungsgraphen, formal definiert. Nach der Änderung einer WF-Instanz ist deshalb ihre problemlose Ausführung sichergestellt.

Das in Abb. 8.1 dargestellte Beispiel dynamischer Änderungen zeigt, dass für diese ggf. ein umfangreicher Umbau des WF-Graphen notwendig wird. In dem in Abb. 8.1a dargestellten WF soll die Aktivität e gelöscht werden. Dies wird realisiert, indem sie durch die leere Aktivität ersetzt wird. Damit diese Änderungsoperation durchgeführt werden darf, muss u.a. überprüft werden, ob von der Aktivität e (erstmalig) geschriebene Datenelemente von nachfolgenden Aktivitäten benötigt werden. Wäre dies der Fall, so könnte das Löschen nicht durchgeführt werden, oder es müssten zusätzliche Maßnahmen stattfinden, um dies zu kompensieren. Etwas aufwendiger als das Löschen ist die in Abb. 8.1 dargestellte Einfügeoperation: Die Aktivität x soll so eingefügt werden, dass sie erst nach der Aktivität c ausgeführt werden kann und beendet sein muss, bevor die Aktivität f gestartet werden kann. Diese Modifikation erfordert das Einfügen einer zusätzlichen parallelen Verzweigung mit den leeren Split- bzw. Join-Knoten n_1 und n_2. Die geforderte Reihenfolgebeziehung zwischen den Aktivität c, x und f wird durch Synchronisationskanten sichergestellt.[1] Das Einfügen der Aktivität wurde so realisiert, dass wieder ein bzgl. des ADEPT-Basismodells gültiger Graph entsteht (inkl. Blockstrukturierung). Die Änderung konnte nur unter der Voraussetzung durchgeführt werden, dass sich ein korrekter Datenfluss ergibt, das heißt z.B., dass die von der Aktivität x gelesenen Datenelemente zuvor geschrieben wurden. Außerdem darf die Aktivität f noch nicht gestartet worden sein, da die Aktivität x sonst nicht vor f bearbeitet werden kann. Um entscheiden zu können, ob eine Änderung durchgeführt werden darf, muss also der gesamte und aktuelle Zustand der betroffenen WF-Instanz bekannt sein. Hier wird auch schon eine Schwierigkeit bei verteilter WF-Ausführung deutlich, da es i.d.R. keinen einzelnen WF-Server gibt, welcher diesen Zustand kennt.

Wenn ein Benutzer eine dynamische Änderung durchführt, so muss das WfMS mit ihm interagieren, um die gewünschte Modifikation zu spezifizieren und ggf. anzupassen (für den Fall, dass sie nicht wie ursprünglich gewünscht durchgeführt werden kann). Deshalb kann eine dynamische Änderung, insgesamt gesehen, recht lange dauern. Sie wird deswegen in zwei Phasen aufgeteilt: In der Phase 1 wird der zu diesem Zeitpunkt gültige Zustand der WF-Instanz ermittelt, mit dem Benutzer interagiert und

[1] Der in Abb. 8.1b dargestellte WF-Graph kann nach Durchführung der Änderungsoperation wieder vereinfacht werden (siehe [Rei00]). Da der Graph auch schon in seiner jetzigen Form die Änderung korrekt repräsentiert, wird hierauf aber nicht eingegangen.

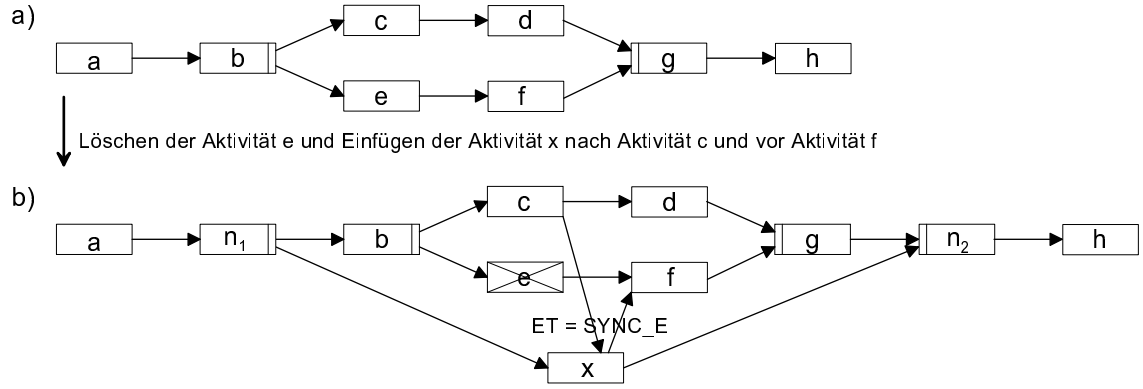


Abbildung 8.1 Beispiel für dynamische Modifikationen.

die Änderung vorbereitet (d.h. die notwendigen Basis-Änderungsoperationen werden berechnet). In der Phase 2 wird der WF-Ausführungsgraph dann tatsächlich umstrukturiert, die Änderungsoperationen werden in der sog. Änderungshistorie der WF-Instanz vermerkt und die Änderungsoperation wird in der Ablaufhistorie registriert. Die Änderung wird aber nur dann durchgeführt, wenn sie aufgrund des aktuellen (und evtl. gegenüber der Phase 1 veränderten) Zustands der WF-Instanz zulässig ist. Damit sich dieser während der Phase 2 nicht verändern kann, wird die WF-Instanz für die Überprüfung der Zulässigkeit und für die Durchführung der Änderung gesperrt. Da in der Phase 2 keine Benutzerinteraktionen stattfinden, ist diese sehr kurz, so dass das Sperren die WF-Ausführung kaum behindert.

8.2 Herausforderungen bei verteilter Workflow-Steuerung

Bei der Entwicklung von ADEPT$_{flex}$ wurde ursprünglich eine logisch zentrale WF-Kontrollinstanz vorausgesetzt. Durch die Kombination mit einer verteilten WF-Steuerung sollen sich keine Einschränkungen für die Funktionalität von dynamischen Änderungen ergeben, d.h. es sollen weiterhin alle zulässigen Änderungen möglich sein und die Konsistenz des entstehenden WF-Ausführungsgraphen muss garantiert werden. Es wird aber zusätzlich angestrebt, die Änderungen möglichst effizient durchzuführen, also unnötigen Kommunikations- und Synchronisationsaufwand zu vermeiden. Um die Verfügbarkeit des WfMS nicht zu beeinträchtigen, dürfen außerdem keine zentralen Komponenten eingesetzt werden. In dem nun folgenden Abschnitt werden die durch diese Forderungen entstehenden Fragestellungen angesprochen, für welche dann im nachfolgenden Abschnitt Lösungen präsentiert werden (siehe auch [RBD99]).

Eine dynamische Änderung kann mehrere, von verschiedenen Servern kontrollierte, Partitionen betreffen. So können z.B. bei der in Abb. 8.1 dargestellten Einfügeoperation die beiden parallelen Zweige von unterschiedlichen Servern kontrolliert werden. Zudem können die Aktivitäten g und h, zwischen denen n_2 eingeführt wird, einem weiteren Server zugeordnet sein. Um die Anwendbarkeit einer Änderungsoperation überprüfen zu können, muss von allen „relevanten“ Servern Zustandsinformation eingeholt werden. Außerdem ist eine Synchronisation mit diesen Servern erforderlich, da eine Änderungsoperation entweder an allen Servern oder überhaupt nicht durchgeführt werden muss.[2] Es stellt sich nun die Frage, welche WF-Server für eine Änderungsoperation relevant sind. Die am einfachsten zu realisierende Lösung ist sicherlich, alle Server des WfMS mit einzubeziehen. Dies

[2] Wenn manche Server den ursprünglichen und manche den geänderten WF-Ausführungsgraphen verwenden würden, so käme es im Zusammenhang mit Migrationen zu Fehlern bei der WF-Steuerung.

führt aber zu einem unnötig hohen Aufwand. Es ist besser, wenn nur diejenigen WF-Server beteiligt werden, die überhaupt jemals (in der Vergangenheit, aktuell oder zukünftig) die entsprechende WF-Instanz steuern. Doch auch diejenigen Server, welche die WF-Instanz nur in der Vergangenheit kontrolliert haben, müssen nicht in die Änderungsoperation einbezogen werden. Mit diesen ist nämlich keine Synchronisation notwendig und die von ihnen verwaltete Zustandsinformation wurde schon durch Migrationen weitergegeben. Im Kontext von variablen Serverzuordnungen kann i.d.R. nicht berechnet werden, welche Server die WF-Instanz zukünftig steuern werden, da die zur Auswertung der Serverzuordnungsausdrücke notwendigen Laufzeitdaten der WF-Instanz evtl. noch nicht existieren. So kann in Abb. 8.2 der Server s_1 den Server der Aktivität h nicht ermitteln, da der Bearbeiter der Aktivität g noch nicht feststeht. Eine Synchronisation mit zukünftigen Servern der WF-Instanz ist also nicht möglich. Eine solche ist auch nicht notwendig, weil diese Server keine Zustandsinformation der WF-Instanz verwalten und keine Synchronisation mit ihnen nötig ist, da sie die WF-Instanz noch nicht kontrollieren. Deshalb bleibt als einzige akzeptable Lösung die Synchronisation mit allen aktuell an der WF-Instanz beteiligten Servern. Es ist aber keineswegs trivial, diese zu ermitteln, weil der Ausführungszustand von parallel ausgeführten Aktivitäten nicht bekannt sein muss. So weiß z.B. in Abb. 8.2 der Server s_1 der Aktivität e nicht, ob die Migration $M_{b,c}$ schon ausgeführt wurde und damit, ob der entsprechende Zweig vom Server s_2 oder s_3 kontrolliert wird. Außerdem ist es nicht ohne weiteres möglich, den für einen parallelen Zweig zuständigen Server zu ermitteln, wenn variable Serverzuordnungen verwendet werden. In Abb. 8.2 referenziert die Serverzuordnung der Aktivität d den Bearbeiter der Aktivität c, der dem Server s_1 nicht bekannt ist.

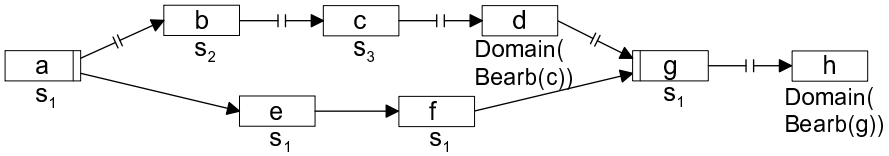


Abbildung 8.2 Dynamische Änderung durch den Server s_1, der gerade die Aktivität e kontrolliert.

Da eine Änderungsoperation nur mit den aktuell an der WF-Instanz beteiligten Servern synchronisiert wird, darf sich diese Menge während (der Phase 2) einer dynamischen Modifikation nicht durch Migrationen verändern. Um dies sicherzustellen, muss die Möglichkeit zur Migration der zugehörigen WF-Instanz vorübergehend gesperrt werden. Da sich während der Phase 2 einer dynamischen Modifikation zusätzlich der Zustand der WF-Instanz nicht verändern darf, wird eine weitere (restriktivere) Art von Sperren notwendig, die Zustandsänderungen verhindert. Die Sperrverfahren müssen so gestaltet werden, dass das Sperren von Migrationen mit dem Sperren von Zustandsänderungen harmoniert.

In eine Änderungsoperation werden nur die aktuell an der WF-Instanz beteiligten Server einbezogen. Deshalb stellt sich die Frage, wie die zukünftig beteiligten Server über diese Änderung informiert werden sollen. Die natürliche Lösung ist, ihnen diese bei der entsprechenden Migration mitzuteilen. Allerdings sollte die zur Übertragung der Änderungsoperation notwendige Datenmenge möglichst minimiert werden (vgl. Kapitel 6). Die Übermittlung der kompletten, neu entstandenen WF-Graphen wäre deshalb äußert ungünstig. Es muss also ein Verfahren entwickelt werden, durch das keine am Migrationszielserver schon bekannte Information (redundant) übertragen wird.

Ein weiteres Problem bei dynamischen Änderungen ergibt sich durch die Verwendung von variablen Serverzuordnungen. Es kann vorkommen, dass eine variable Serverzuordnung nach einer Änderung nicht mehr ausgewertet werden kann, weil die referenzierte Aktivität gelöscht wurde (z.B. für die

Aktivität d in Abb. 8.2 nach einem Löschen der Aktivität c). Für diesen Fall wird ein Verfahren benötigt, mit dem trotzdem ein gut geeigneter Server für diese Aktivität festgelegt werden kann.

Auch die Serverzuordnungen der von einer dynamischen Modifikation direkt betroffenen Aktivitäten stellen ein Problem dar: Bei neu eingefügten Aktivitäten ist offensichtlich, dass für diese noch eine (geeignete) Serverzuordnung festgelegt werden muss. Doch auch die Serverzuordnungen von gelöschten Aktivitäten (die also durch leere Aktivitäten ersetzt wurden) müssen betrachtet werden. Es macht sicherlich keinen Sinn, eine Migration zu einem Server durchzuführen, dessen Partition aus lauter leeren Aktivitäten besteht. Da die Ausführung dieser Aktivitäten keine Kommunikation erfordert, können sie ebenso gut von einem anderen Server kontrolliert werden, so dass die entsprechende Migration eingespart wird. Deshalb sollten bei einer dynamischen Änderung die Serverzuordnungen gelöschter Aktivitäten ebenfalls überarbeitet werden.

8.3 Dynamische Änderungen in ADEPT$_{distribution}$

Dieser Abschnitt gibt Antworten auf die soeben aufgeführten Fragestellungen. Dabei behandelt der erste Unterabschnitt Probleme, die durch die erforderliche Synchronisation der aktuell an einer WF-Instanz beteiligten Server entstehen, und der zweite Fragestellungen, die sich durch nicht mehr auswertbare bzw. noch festzulegende Serverzuordnungen ergeben.

8.3.1 Synchronisation der Workflow-Server

Wir haben schon festgestellt, dass für die Durchführung einer dynamischen Änderung die aktuell an der betroffenen WF-Instanz beteiligten Server ermittelt werden müssen, um Zustandsinformation auszutauschen und um die Änderung zu synchronisieren. Die Menge dieser Server könnte bestimmt werden, indem per Broadcast bei allen Servern des WfMS nachgefragt wird, ob sie an dieser Instanz beteiligt sind. Ein solches Verfahren ist aber (besonders in weiträumig verteilten Systemen) zu aufwendig. Außerdem können dann keine dynamischen Änderungen durchgeführt werden, solange nicht alle Server erreichbar sind. Deshalb wurde ein Verfahren entwickelt, bei dem ein Koordinator die an einer WF-Instanz beteiligten Server verwaltet. Als Koordinator wird der Startserver der WF-Instanz verwendet, da dieser allen an der Instanz beteiligten Servern aus der Ablaufhistorie bekannt ist und dennoch für das WfMS keine zentrale Komponente darstellt. Diesem Koordinator wird bei jeder Migration gemeldet, ob ein Server für diese WF-Instanz hinzukommt bzw. wegfällt und welche Server dies sind. Wenn nun eine dynamische Änderung durchgeführt werden soll, so wendet sich der entsprechende Server an den Koordinator, um die Menge der an dieser WF-Instanz beteiligten Server zu erfragen.

Nachdem der Koordinator die Menge der an der zu modifizierenden WF-Instanz beteiligten Server übermittelt hat, darf sich diese Menge nicht verändern, bevor von diesen die Zustandsinformation zur WF-Instanz eingeholt wurde bzw. die Änderungsoperation zwischen diesen Servern synchronisiert wurde. Ansonsten könnte es vorkommen, dass Server nicht in eine Änderungsoperation einbezogen werden, obwohl sie die entsprechende WF-Instanz gerade kontrollieren, was zu einer fehlerhaften Ausführung der Änderungsoperation führen würde. Um dies zu verhindern, wird beim Koordinator eine Sperre gesetzt, welche durchzuführende Migrationen blockiert. Damit dies funktioniert, müssen die WF-Server vor der Durchführung einer Migration beim Koordinator ebenfalls eine Sperre anfordern. Diese Sperre wird nicht gewährt, falls eine dynamische Änderungsoperation gerade eine Sperre

hält. Ebenso sind zeitlich überlappende Modifikationen derselben WF-Instanz ausgeschlossen. Für den Regelfall, dass gerade keine dynamische Änderung stattfindet, soll es aber möglich sein, mehrere Migrationen derselben WF-Instanz überlappend durchzuführen. Deshalb müssen die von Migrationen angeforderten Sperren untereinander verträglich sein. Eine von einer Migration angeforderte Sperre entspricht also einer Lesesperre bei einem DBMS und eine von einer Änderungsoperation gehaltene Sperre stellt eine Schreibsperre (exklusive Sperre) dar (vgl. [GR93, HR99]). Das vollständige Protokoll für die Verwaltung der an einer WF-Instanz beteiligten Server und der Sperren ist in [Zei99] dargestellt.

Nachdem eine dynamische Änderung von allen aktuell an der betroffenen WF-Instanz beteiligten Servern durchgeführt wurde, kann ihre Bearbeitung fortgesetzt werden. Dies kann erfordern, dass zu einem WF-Server migriert wird, dem diese und vorangehende Änderungsoperationen noch nicht bekannt sind. Doch auch ein solcher Server benötigt zur WF-Ausführung den aktuellen Ausführungsgraphen. Diesen Graphen bei der Migration komplett zu übertragen, würde einen großen und zudem unnötigen Aufwand bedeuten. Der Zielserver kennt schließlich schon die Vorlage der WF-Instanz, ihm müssen also nur noch die Änderungen bekannt gemacht werden. Zu diesem Zweck wird ihm die (ohnehin benötigte [Rei00]) Änderungshistorie übertragen, so dass er die in ihr vermerkten Modifikationen auf den ursprünglichen Graphen anwenden kann. Dadurch erhält er den aktuellen Ausführungsgraphen der WF-Instanz. Doch auch Teile der Änderungshistorie können dem Zielserver schon bekannt sein, nämlich dann, wenn er die WF-Instanz in der Vergangenheit schon einmal kontrolliert hat. Deshalb genügt es, den noch benötigten Teil der Änderungshistorie zu übertragen. Hierfür bietet sich (ebenso wie für die Ablaufhistorie und die großen Datenelemente in Kapitel 6) ein Verfahren an, bei dem die entsprechenden Einträge angefordert werden, weil nur so redundante Übertragungen verhindert werden können. Das resultierende Verfahren ist vergleichsweise einfach, da stets jeder an einer WF-Instanz beteiligte Server alle bis zu diesem Zeitpunkt erfolgten Änderungsoperationen kennt. Deshalb ist einem ehemals an der WF-Instanz beteiligten Server der „Anfang“ der Änderungshistorie bekannt, d.h. er verfügt bis zu einer bestimmten Stelle über alle Einträge und ab dieser Stelle kennt er keine weiteren Einträge. Zum Anfordern der fehlenden Einträge der Änderungshistorie genügt es also, die ID des letzten bekannten Eintrags an den Quellserver der Migration zu übertragen, worauf dieser alle auf diesen Eintrag folgenden Einträge übermittelt. Das Anfordern und Übertragen der Änderungshistorie erfordert nicht einmal einen zusätzlichen Kommunikationszyklus, da dies gemeinsam mit dem Anfordern und Übertragen der Ablaufhistorie (siehe Abschnitt 6.2.2) erfolgen kann.

8.3.2 Serverzuordnungen und dynamische Änderungen

Wird die in $ServZuordn_n$ referenzierte Aktivität $m = ReferencedAct_n$ gelöscht, so wird in $ServZuordn_n$ ein Laufzeitdatum der WF-Instanz verwendet, das nicht existiert. So entsteht durch das in Abb. 8.3 dargestellte Löschen der Aktivität d ein Problem für $ServZuordn_f$ = `Domain(Bearb(d))`. Es wäre nun möglich, dies zu verhindern, indem dieselben Verfahren eingesetzt werden, mit denen auch die Parameterversorgung der Aktivitäten sichergestellt wird (siehe [Rei00]). Dies würde aber verhindern, dass die Aktivität d gelöscht werden kann, obwohl dies im zentralen Fall möglich ist. Diese Restriktion kann vermieden werden, indem im Falle einer nicht auflösbaren Serverzuordnung einfach ein beliebiger Server gewählt wird. Durch eine solche Vorgehensweise kann aber ein äußerst ungünstiger WF-Server verwendet werden. Das folgende Verfahren vermeidet Einschränkungen für die Anwendbarkeit von dynamischen Änderungen und beeinträchtigt zudem die Qualität der gewählten WF-Server nicht: Es ist stets möglich, den ursprünglich für eine gelöschte Aktivität m vorgesehenen Serverzuordnungsausdruck auszuwerten, so dass der Server ermittelt wer-

den kann, der eigentlich für die Aktivität m vorgesehen war. Im Falle vom $ServZuordn_n$ = `Server(m)` bzw. $ServZuordn_n$ = `f(Server(m))` kann dieser referenziert werden, womit sich für die Aktivität n der ursprünglich geplante WF-Server ergibt. Solche Serverzuordnungen stellen also kein Problem dar. So wird in dem in Abb. 8.3b dargestellten Beispiel für die Aktivität g der Server s_3 verwendet, da dieser auch für die Aktivität d vorgesehen war (vgl. Abb. 8.3a). Kritischer ist der Fall, dass in $ServZuordn_n$ der Bearbeiter einer gelöschten Aktivität m referenziert wird. Da die Aktivität m durch eine leere Aktivität ersetzt wurde, wird sie von niemandem bearbeitet. Das Problem lässt sich dadurch lösen, dass gelöschten Aktivitäten ein fiktiver Bearbeiter zugeordnet wird. Dieser wird ermittelt, indem aus der Menge der Benutzer, welche die Aktivität m aufgrund von $BearbZuordn_m$ bearbeiten dürfen, einer zufällig[3] ausgewählt wird. Dieser erfüllt alle Kriterien, die an den Bearbeiter der Aktivität m gestellt werden, so dass er mit einer ebenso großen Wahrscheinlichkeit einem günstigen Domain angehört, wie der tatsächliche Bearbeiter einer Aktivität m. Deshalb ergibt sich bei Serverzuordnungen der Art $ServZuordn_n$ = `Domain(Bearb(m))` bzw. $ServZuordn_n$ = `f(Domain(Bearb(m)))` ein ebenso günstiger WF-Server. Im Beispiel aus Abb. 8.3 wird also beim „Durchschalten“ der Aktivität d ein fiktiver Bearbeiter u ausgewählt, welcher aufgrund von $BearbZuordn_d$ die Aktivität d hätte bearbeiten dürfen. Dessen Domain wird dann für den Server der Aktivität f gewählt.

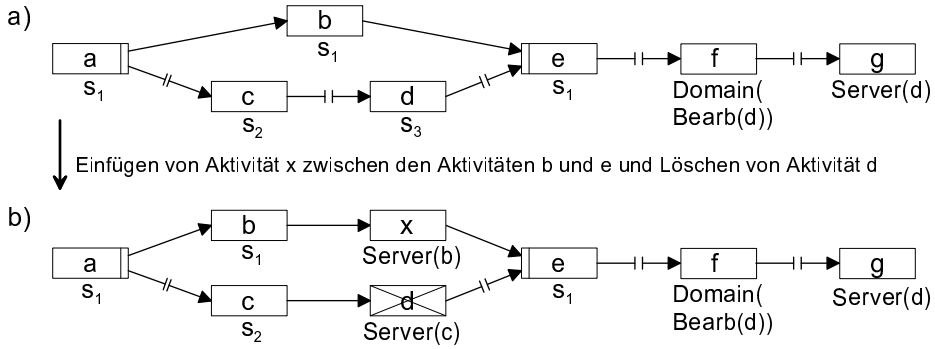


Abbildung 8.3 Auswirkungen von dynamischen Änderungen auf Serverzuordnungen.

Wird eine Aktivität zur Ausführungszeit neu in einen WF eingefügt, so muss ihr ein Server zugeordnet werden. Den entsprechenden Serverzuordnungsausdruck mit den für die Modellierungszeit vorgesehenen Verteilungsalgorithmen zu berechnen, würde einen zu großen Aufwand erfordern. Wird der Server dieser Aktivität (ohne Strategie) beliebig festgelegt, so ergibt sich ein sehr ungünstiges Kommunikationsverhalten, weil zusätzliche Migrationen erforderlich werden. Deshalb wird ein einfaches Verfahren benötigt, mit dem eine „recht gute Serverzuordnung“ bestimmt werden kann. Das entwickelte Verfahren basiert auf der folgenden Beobachtung: Besonders hohe Kommunikationskosten resultieren aus unnötigen Migrationen. Deshalb ist es eine vernünftige Strategie, diese zu vermeiden, indem eine neu eingefügte Aktivität einer angrenzenden Partition zugeordnet wird. Da es im Kontext von variablen Serverzuordnungen kaum möglich ist, die neue Aktivität n einer nachfolgenden Partition zuzuordnen (vgl. Abschnitt 5.1.5.3), wird zu diesem Zweck die Partition einer beliebigen Vorgängeraktivität $m \in Pred_{\{CONTROL\_E\}}(n)$ ausgewählt. Durch Verwendung der Serverzuordnung $ServZuordn_n$ = `Server(m)` wird die Aktivität n dieser Partition zugeordnet. Die in Abb. 8.3b neu hinzugekommene Aktivität x erhält also $ServZuordn_x$ = `Server(b)`, so dass zwischen den Aktivitäten b und x keine Migration stattfindet.

[3] Bei dieser „zufälligen Auswahl“ sollten die Gewichte $G_n(u)$ der Bearbeiter berücksichtigt werden, damit die Wahrscheinlichkeit des fiktiven Bearbeiters der eines tatsächlichen Bearbeiters entspricht.

Auch beim Löschen von Aktivitäten müssen die Serverzuordnungen betrachtet werden: Besteht eine Partition ausschließlich aus leeren Aktivitäten, so macht es keinen Sinn, für die Ausführung dieser Partition zu einem anderen Server zu migrieren, weil die Ausführung leerer Aktivitäten keine Kommunikation erfordert. Deshalb sollten auch die Serverzuordnungen einer gelöschten (d.h. durch eine leere Aktivität ersetzten) Aktivität überarbeitet werden. Diese sollte, ebenso wie für neu eingefügte Aktivitäten beschrieben, einer Vorgängerpartition zugeordnet werden. So erhält die in Abb. 8.3 gelöschte Aktivität d die Serverzuordnung $ServZuordn_d = \texttt{Server(c)}$, so dass die Migration zwischen den Aktivitäten c und d entfällt. Beim Löschen ist zudem zu beachten, dass die Aktivitäten einer Partition durch mehrere hintereinander ausgeführte Operationen gelöscht werden können, und dass mehrere aufeinander folgende Partitionen gelöscht werden können. Darum sollte das beschriebene Verfahren auch auf nachfolgende leere Aktivitäten angewendet werden. Wird z.B. in dem in Abb. 8.3b dargestellten WF zusätzlich die Aktivität c gelöscht, so wird – wie oben beschrieben – ihre Serverzuordnung in $ServZuordn_c = \texttt{Server(a)}$ verändert. Die Aktivität d würde aber von dem ursprünglich für die Aktivität c vorgesehenen Server s_2 kontrolliert werden, was zu einer Migration zwischen den Aktivitäten c und d führen würde. Deshalb wird von dem Verfahren auch die Serverzuordnung der auf die Aktivität c folgenden leeren Aktivität d in $ServZuordn_d = \texttt{Server(a)}$ verändert, so dass durch das Löschen der Aktivität c insgesamt zwei Migrationen entfallen.

8.4 Zusammenfassung und Ausblick

Um WfMS in der Praxis für ein breites Spektrum von Anwendungen einsetzen zu können, ist es erforderlich, dass zur Ausführungszeit der WF-Instanzen dynamisch von ihrem Ablauf abgewichen werden kann. Damit dabei weiterhin eine sichere WF-Ausführung garantiert ist, muss zuvor die Anwendbarkeit einer solchen Änderungsoperation geprüft werden. Um dies zu ermöglichen, wird der globale aktuelle Zustand der WF-Instanz benötigt. Dieser ist aber in ADEPT$_{distribution}$ keinem (einzelnen) WF-Server bekannt. In diesem Kapitel ist es trotz der widersprüchlichen Anforderungen und Ziele von dynamischen Änderungen und effizientem verteilten WF-Management gelungen, ein Verfahren zu entwickeln, das weiterhin alle zulässigen Änderungsoperationen ermöglicht und dennoch keinen hohen Kommunikationsaufwand erfordert. Insbesondere behindert es die „normale“ verteilte WF-Ausführung nicht.

Die vorgestellten Verfahren stellen eine praktikable Möglichkeit zur Kombination von verteilter WF-Steuerung und dynamischen Änderungen dar. Sie können allerdings noch erweitert werden, wobei aber die Relevanz der im Folgenden vorgestellten Erweiterungen für die Praxis noch zu klären ist. Um die Serverzuordnungen neu eingefügter Aktivitäten festzulegen, könnte man sich intelligente Verfahren vorstellen, welche die Verteilung der Bearbeiter dieser Aktivitäten auf die Teilnetze einbeziehen. So wäre es auch für eine neu eingefügte Aktivität möglich, einen Server festzulegen, der sich nahe bei den potentiellen Bearbeitern der Aktivität befindet. Um die Festlegung der Serverzuordnungen effizient zu gestalten, kann hierfür zur Modellierungszeit vorberechnete Information genutzt werden. Außerdem kann es sinnvoll sein, auch die Serverzuordnungen benachbarter Aktivitäten zu überarbeiten, um ggf. Migrationen einzusparen. Da dynamische Änderungen aber verhältnismäßig seltene Operationen sind, ist fraglich, ob solche Optimierungen tatsächlich zu einer merklichen Reduktion der Kommunikationskosten führen.

Ein anderer Ansatzpunkt für Optimierungen sind Änderungsoperationen, deren Auswirkungen auf einen kleinen Teil des WF-Graphen begrenzt sind (z.B. das Löschen von Aktivitäten in nur einem Zweig einer Parallelität). In diesen Fällen würde es genügen, nur manche Server der WF-Instanz in die

Änderungsoperation einzubeziehen (d.h. nur bei diesen Servern wird Zustandsinformation angefordert und die Änderung durchgeführt). Im günstigsten Fall kann eine solche Änderungsoperation von einem einzigen Server, ohne Kommunikation mit den Servern parallel ausgeführter Aktivitäten, durchgeführt werden. Dies ist dann der Fall, wenn die Änderung nur auf Aktivitäten dieses Servers Auswirkungen hat. Allerdings haben unsere Untersuchungen [Zei99] ergeben, dass selbst Änderungsoperationen, die nur begrenzte Umstrukturierungen des WF-Graphen erfordern, durch Abhängigkeiten im Datenfluss weitreichende Auswirkungen haben können. Deshalb kann eine solche Optimierung nur in extrem seltenen Ausnahmefällen eingesetzt werden.

Teil III

Diskussion und Zusammenfassung

Kapitel 9

Klassifikation und Simulation von Workflow-Management-Systemen

Nachdem im Teil II das Verteilungsmodell von ADEPT vorgestellt wurde, soll es in diesem Kapitel mit anderen möglichen Verteilungsmodellen verglichen werden. Zu diesem Zweck werden aus der Literatur bekannte Verteilungsmodelle klassifiziert. Auch ADEPT$_{distribution}$ wird in diese Klassifikation eingeordnet und mit den anderen Verteilungsmodellen verglichen. Dieser Vergleich wird dann durch quantitative Ergebnisse ergänzt. Diese erhalten wir durch Simulationen, die auf den zuvor ermittelten Klassen basieren. Das heißt, für mehrere Szenarien wird das Verhalten von WfMS der jeweiligen Klassen bei der WF-Ausführung simuliert. Die Ergebnisse dieser Simulationen werden dann vorgestellt und diskutiert. Die Klassifikation der WfMS-Verteilungsmodelle und einige der in diesem Kapitel präsentierten Simulationen wurden auch schon in [BD99a, BD99b, BD00a] publiziert.

Die eigentliche Diskussion der originären Beiträge dieser Arbeit erfolgt dann in Kapitel 10, wobei auf die im vorliegenden Kapitel vorgestellten Verteilungsmodelle und die Simulationsergebnisse Bezug genommen wird.

In Abschnitt 9.1 wird erläutert, anhand welcher Kriterien die Verteilungsmodelle klassifiziert werden, die verschiedenen Verteilungsmodelle werden vorgestellt und ihnen werden konkrete Systeme und Forschungsansätze zugeordnet. Außerdem wird eine geeignete Vorgehensweise für die Auswahl eines Verteilungsmodells für eine gegebene Anwendung erläutert. In den nachfolgenden Abschnitten wenden wir uns dann den Simulationen zu. In Abschnitt 9.2 werden die Anforderungen an die Simulationsumgebung und deren Entwurf beschrieben. Abschnitt 9.3 erläutert das Vorgehen bei der Auswertung der Simulationsdaten. In den darauf folgenden drei Abschnitten werden Simulationsszenarien vorgestellt und die zugehörigen Simulationsergebnisse präsentiert und interpretiert: Für die Simulation aus Abschnitt 9.4 wird ein möglichst realitätsnaher WF verwendet. Dies erlaubt zwar keine besonders exakte Interpretation der Auswirkungen der einzelnen Aspekte (z.B. variable Serverzuordnungen), dafür kommen die Vor- und Nachteile aller Verteilungsmodelle zur Geltung. In Abschnitt 9.5 wird ein WF simuliert, bei dem keine abhängigen Bearbeiterzuordnungen verwendet werden. In Abschnitt 9.6 wird die Simulation eines WF vorgestellt, der ein hohes Maß an abhängigen Bearbeiterzuordnungen aufweist, so dass die Auswirkungen der dann verwendbaren variablen Serverzuordnungen beurteilt werden können. Das Kapitel schließt mit einer Zusammenfassung der gewonnenen Erkenntnisse in Abschnitt 9.7.

9.1 Verteilungsmodelle für Workflow-Management-Systeme

Dieser Abschnitt bietet eine Klassifikation von aus der Literatur bekannten Verteilungsmodellen für WfMS. Zuerst wird geklärt, aufgrund welcher Kriterien die WfMS klassifiziert werden sollen. Anschließend findet eine Einordnung von konkreten Systemen und Forschungsansätzen in die Klassifikation statt. Der Abschnitt endet damit, dass erläutert wird, wie für ein konkretes Szenario ein geeignetes Verteilungsmodell ausgewählt werden kann.

9.1.1 Vorgehensweise beim Vergleich der Verteilungsmodelle

Zum Vergleich der Verteilungsmodelle betrachten wir die Ausführung (eines Ausschnitts) eines WF, der aus der Sequenz der Aktivitäten A und B besteht. In Abb. 9.1 sind die hinsichtlich der späteren Diskussion interessanten Aktionen aufgeführt. Der WF-Server ermittelt die potentiellen Bearbeiter von Aktivität A und fügt Aktivität A in deren Arbeitslisten ein. Ein Benutzer, der diese Aktivität bearbeiten möchte, wählt sie aus seiner Arbeitsliste aus; der WF-Server synchronisiert diesen Vorgang (A: AL). Anschließend startet der WF-Server oder Client das entsprechende Aktivitätenprogramm für diesen Benutzer (A: Ausf). Dies findet i.d.R. zumindest für den Client-Teil des Programms auf dem Rechner dieses Bearbeiters statt. Zum Starten müssen die Eingabedaten zu dem Programm übertragen werden. Nach Beendigung des Programms und der Übertragung der Ausgabedaten zum Server erfolgt derselbe Ablauf für Aktivität B (B: AL und B: Ausf). Da jede der in Abb. 9.1 dargestellten Aktionen potentiell auf einem anderen Rechner ausgeführt werden kann, stellen die Pfeile 1-7 mögliche Kommunikationen dar. Im Abschnitt 1.2.2 wurde gezeigt, dass auch das dem WfMS zugrunde liegende Kommunikationssystem zum Flaschenhals werden kann. Deshalb wird im Folgenden nicht nur die Belastung der WF-Server, sondern auch die des Kommunikationssystems betrachtet.

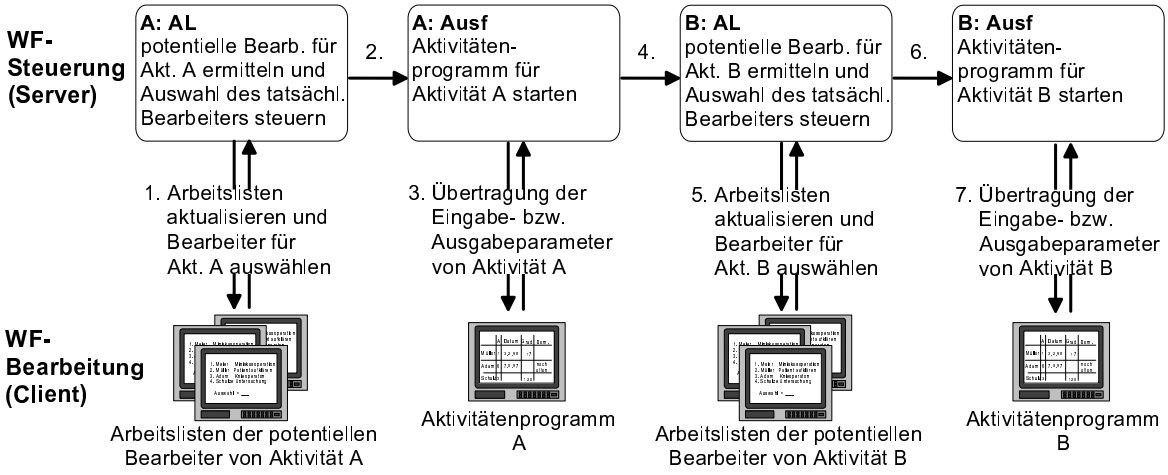


Abbildung 9.1 Steuerung und Bearbeitung eines WF, der aus den Aktivitäten A und B besteht.

Skalierbarkeit und hohe Verfügbarkeit von WfMS wird durch Replikation von Systemkomponenten und ihre geeignete Verteilung auf mehrere Rechner und Teilnetze erreicht. Die dafür möglichen Verteilungsmodelle werden in diesem Kapitel beschrieben, klassifiziert und analysiert und ihnen konkrete kommerzielle Systeme sowie Vorschläge aus dem Bereich der Forschung zugeordnet. Die Konzepte werden hierbei bezüglich ihres Leistungsverhaltens miteinander verglichen. Dazu wird untersucht, wie sich die Last für die Steuerung der Aktivitäten und das Aktualisieren der Arbeitslisten in der

Belastung der Server und der Teilnetze niederschlägt. In einem unternehmensweiten WfMS können diese Werte sehr groß werden. Damit keine Komponente überlastet wird, muss ihre Belastung unter die jeweils vorgegebene Maximalkapazität der Komponente gedrückt werden können. Dies ist z.B. dann gegeben, wenn die Last für jede Komponente $O(\text{Gesamtlast}/x)$ beträgt, wobei x eine konfigurierbare Größe sein muss, wie etwa die Anzahl der Server-Replikate im System. Ist diese Bedingung durch ein Verteilungsmodell erfüllt, so kann dieses als skalierbar bezeichnet werden.

Abb. 9.2 zeigt eine Kategorisierung der Verteilungsmodelle und die Einordnung konkreter Systeme. Die beiden Extrempunkte des Spektrums sind Systeme mit nur einem zentralen Server und voll verteilte Systeme, bei denen jede Komponente Serveraufgaben wahrnimmt. Dazwischen liegen Systeme, die mehr als einen Server verwenden, aber trotzdem noch die Trennung zwischen Client und Server aufrechterhalten, so dass es weniger Server als Clients (Benutzerrechner) im System gibt. Die Übergänge zwischen den Klassen sind fließend, und natürlich gibt es Systeme, die Mischformen darstellen. Im Folgenden werden die Verteilungsmodelle aber in ihrer Reinform dargestellt, um die Unterschiede deutlich herausarbeiten zu können.

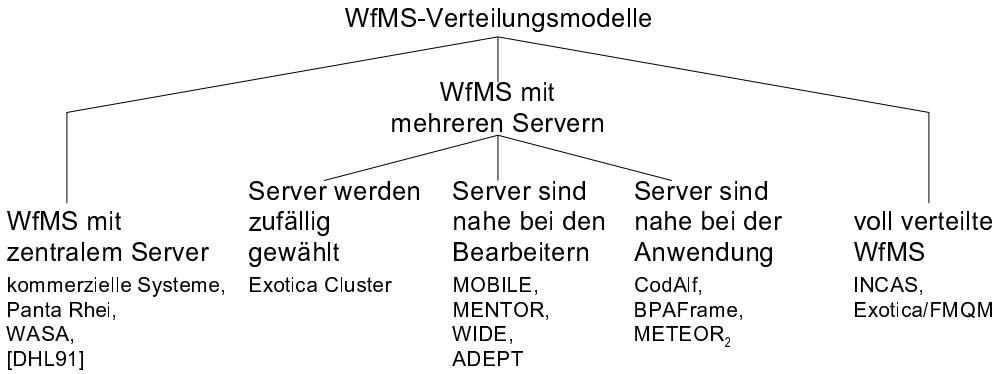


Abbildung 9.2 Verteilungsmodelle für WfMS und Einordnung entsprechender Systeme.

9.1.2 Workflow-Management-Systeme mit zentralem Server

Systeme dieser Kategorie verwenden eine zentrale Serverkomponente (vgl. Abb. 9.3). Dies ist zumindest eine zentrale WF-DB mit nur einem DB-Server. Meist wird auch nur ein WF-Server verwendet, es gibt aber auch Systeme, bei denen sich mehrere WF-Server die zentrale DB teilen.

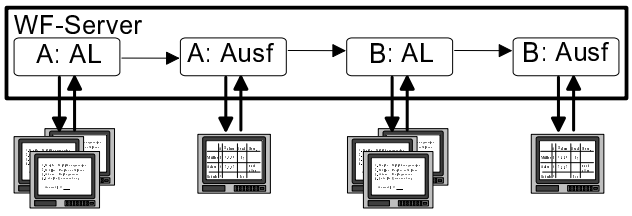


Abbildung 9.3 Ein zentraler WF-Server bearbeitet alle WF-Instanzen des WfMS.

Die Leistungsfähigkeit eines solchen Systems ist im Wesentlichen durch den zentralen Server bestimmt bzw. beschränkt, da der Server die volle Last für die Ausführung der Aktivitäten und für die Aktualisierung der Arbeitslisten zu tragen hat. Entsprechend stark wird das Teilnetz des DB-Servers belastet. Ein solches System ist deshalb nur für eine relativ kleine Anzahl von Benutzern (einige

Dutzend) geeignet. Werden mehrere WF-Server verwendet, so lässt sich die Last zwischen ihnen aufteilen, das zentrale DBMS bleibt aber weiterhin ein potentieller Flaschenhals.

Beispiele:
In die Kategorie „zentraler Server“ fallen viele kommerzielle Systeme wie **WorkParty** [Sie95a], **ProMInanD** [Kar94] oder **FlowMark** [IBM96b] bis Version 2.1. Sie verwenden einen zentralen WF-Server oder zumindest eine zentrale WF-DB (mit zentralem DB-Server), die zur Ausführung jedes Teilschritts benötigt wird. Auch einige Forschungsansätze (z.B. **Panta Rhei** [EG96, EGL96a, EGL96b, EL96a, EL96b], **WASA** [VWW96a, VWW96b], **[DHL91]**) setzen eine zentrale Kontrollinstanz voraus.

FlowMark [IBM96a, IBM96b] ermöglicht ab Version 2.2, einen Subprozess in einem anderen FlowMark-System (Local Domain) ausführen zu lassen. Dadurch entsteht eine Mischform aus einer zentralen Architektur und einer Architektur, bei der mehrere Server verwendet werden, die jeweils nahe bei den jeweiligen Bearbeitern angesiedelt sind (siehe Abschnitt 9.1.3.2). Eine solche Kooperation von zentralen WfMS ist nützlich, wenn ein Teil eines Prozesses von anderen Bearbeitern bzw. in anderen OE ausgeführt wird als der Rest. Dieser Teil wird dann von dem anderen WfMS kontrolliert. Es ist hierbei zu beachten, dass es sich bei den verschiedenen Local Domains um getrennte Systeme handelt, die lediglich kooperieren. Dies zeigt sich zum Beispiel daran, dass Benutzer, die Aktivitäten in verschiedenen Abschnitten eines solchen aufgespaltenen Prozesses ausführen sollen, in allen zugehörigen Systemen bekannt gemacht werden müssen.

Im Optimalfall verteilt sich die Last gleichmäßig auf alle beteiligten (zentralen) Systeme. Auch der Aufwand für das Aktualisieren der Arbeitslisten wird aufgeteilt, wenn die Benutzer jeweils nur in wenigen Local Domains aktiv und gleichmäßig auf sie verteilt sind. Der Aufwand für das Ändern einer Verteilung ist allerdings sehr groß, da entsprechende Teilprozesse für die einzelnen Teilsysteme modelliert werden müssen, d.h. das WF-Modell verändert werden muss.

Diskussion:
Bei einem zentralen WF-Server stößt man sehr schnell an dessen Leistungsgrenze und an die des betroffenen Kommunikationssystems. Wie in Abschnitt 3.1.2 diskutiert wurde, führt auch der Einsatz eines (zentralen) Hochleistungssystems nicht automatisch zur Lösung dieser Probleme. Werden, wie bei FlowMark, kooperierende zentrale WfMS verwendet, so wird zwar ein verteiltes WF-Management ermöglicht, die dabei realisierte Verteilung ist aber sehr unflexibel (vgl. Abschnitt 3.4.2).

9.1.3 Workflow-Management-Systeme mit mehreren Servern

Systeme, bei denen der WF-Server mehrfach und auf verschiedene Maschinen verteilt vorhanden ist, bilden die im Folgenden beschriebene Kategorie. Für die Verteilung der Server gibt es drei Ansätze: Man wählt den WF-Server nach dem Zufallsprinzip oder man versucht, aus dem Ort des Servers Vorteile zu ziehen. Hier gibt es zwei Möglichkeiten: Man kann für die Steuerung einer Aktivität den Server verwenden, der sich bei den vorgesehenen Bearbeitern befindet oder den, der sich auf dem Knoten der zugehörigen Anwendung befindet.

9.1.3.1 Server werden zufällig gewählt

Bei diesem Ansatz wird der Server für eine WF-Instanz zufällig gewählt (Abb. 9.4). Die Server sind identische Replikate der WF-Engine und bestehen aus einer WF-DB und einem WF-Server (WF-

Cluster). Alle WF-Typen können von jedem Cluster ausgeführt werden; eine WF-Instanz verbleibt in dem Cluster, in dem sie gestartet wurde. Ein Cluster ist nicht an eine OE gebunden.

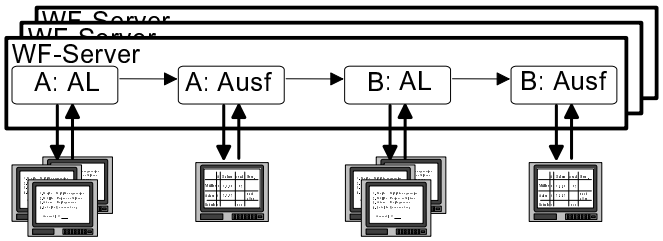


Abbildung 9.4 Der Cluster für eine WF-Instanz wird bei deren Start zufällig gewählt.

In der WF-DB sind die replizierte Schemainformation, die Instanzen des Clusters und der diesen Cluster betreffende Teil der Arbeitslisten aller Benutzer gespeichert. Jeder Client muss eine Verbindung mit jedem Cluster aufbauen, da in jedem Cluster Aufträge für ihn vorhanden sein können.

Durch die geeignete Wahl des Clusters beim Prozessstart (zufällig, zyklisch, lastabhängig) kann eine Lastbalancierung erreicht werden. Dadurch lässt sich die durch die Aktivitätenausführung entstehende Last (Parametertransfer zum Client) auf die WF-DB verteilen. Der Ausfall einer WF-DB blockiert zwar alle Instanzen des zugehörigen Clusters, die WF-Instanzen anderer Cluster sind hiervon aber nicht betroffen. Da die Benutzer normalerweise auch WF-Instanzen anderer Cluster bearbeiten, können i.d.R. alle Benutzer, trotz des Ausfalls eines Clusters, zumindest eingeschränkt weiterarbeiten.

Für das Aktualisieren der Arbeitslisten stellen die Cluster ein potentielles Problem dar. Von einem Cluster werden jeweils Instanzen aller WF-Typen kontrolliert, weshalb alle Benutzer des WfMS für diesen Cluster relevant sind. Deshalb muss ein Cluster Teile der Arbeitslisten aller Benutzer verwalten, weshalb er einen potentiellen Flaschenhals bildet. Die Anzahl der zu verwaltenden Arbeitslisten kann also durch die Verwendung mehrerer Cluster nicht verringert werden, was bei sehr großen Benutzerzahlen ein Problem sein kann.

Ein anderes Problem ergibt sich durch die Belastung des Netzwerks. Durch die Erhöhung der Anzahl der Teilnetze (auf welche die Cluster verteilt werden) lässt sich zwar ihre Kommunikationslast senken, aber durch die zufällige Wahl des Clusters beim Start einer Instanz werden auch Cluster mit ungünstiger Lage gewählt. Dieses Problem ist bei weiträumig verteilten Systemen besonders gravierend. So kann die Bearbeitung eines Prozesses zwar geographisch beschränkt sein, aber durch die unglückliche Wahl des Clusters für eine Instanz ständige WAN-Kommunikation zur Steuerung notwendig werden.

Beispiel:
Beim **Exotica-Cluster**-Ansatz [AKA+94] bestehen die Cluster aus je einer WF-DB und mehreren WF-Servern. Ein Client muss sich mit nur einem WF-Server jedes Clusters verbinden, da durch die gemeinsame WF-DB alle Server eines Clusters über alle notwendigen Informationen verfügen.

Die Verwendung mehrerer WF-Server in einem Cluster erhöht die Verfügbarkeit, da ein Client beim Ausfall eines WF-Servers einen anderen Server dieses Clusters verwenden kann. Der Ausfall der WF-DB blockiert natürlich weiterhin den gesamten betroffenen Cluster. Die Verwendung mehrerer WF-Server je Cluster reduziert die Last pro Server, die durch die Aktivitätenausführung und das Aktualisieren der Arbeitslisten entsteht. Da die Server eines Clusters alle Benutzer bedienen müssen, bildet die WF-DB beim Aktualisieren der Arbeitslisten aber weiterhin einen Flaschenhals.

Diskussion:
Im Gegensatz zu ADEPT$_{distribution}$ weist dieses Verteilungsmodell ein sehr ungünstiges Kommunikationsverhalten auf, da der Ort des WF-Servers nicht gezielt gewählt werden kann. Außerdem muss

der Server jedes Clusters die Arbeitslisten aller Benutzer aktualisieren, während ein WF-Server in ADEPT nur die Bearbeiter bestimmter (ihm zugeordneter) Aktivitäten bedienen muss.

9.1.3.2 Server sind nahe bei den Bearbeitern

Die im Folgenden beschriebenen Ansätze versuchen aus der gezielten Wahl des WF-Servers Vorteile zu ziehen. Dazu wird jeweils der WF-Server verwendet, der nahe bei den Bearbeitern liegt. Dieser Ansatz basiert auf der Annahme, dass die meisten Aktivitäten von Benutzern ausgeführt werden, die (fast) alle derselben OE angehören. Diese stellt somit einen guten Ort für den WF-Server dar. Eine OE kann jede Art von organisatorischer Einheit sein. Allerdings wird in diesem Zusammenhang gefordert, dass die OE geographisch beschränkt ist.

Die Konzepte dieser Kategorie können danach unterschieden werden, ob eine WF-Instanz den Server wechseln kann, also eine Migration möglich ist (Abb. 9.5b), oder nicht (Abb. 9.5a). Dass sich Migrationen auf das Kommunikationsverhalten eines WfMS positiv auswirken können, wurde schon in den Abschnitten 3.2.2 und 4.7.2.2 gezeigt. Bei den Ansätzen, bei denen keine Migration möglich ist, wird von der Annahme ausgegangen, dass ein kompletter WF weitgehend zu nur einer OE gehört. Aktivitäten, die in anderen OE ausgeführt werden, werden ebenfalls von diesem, dann natürlich nicht optimalen Server gesteuert. Wird die Anzahl dieser „entfernten“ Aktivitäten zu groß, so kann die durch ihre Ausführung erzeugte Kommunikationslast sehr hoch werden.

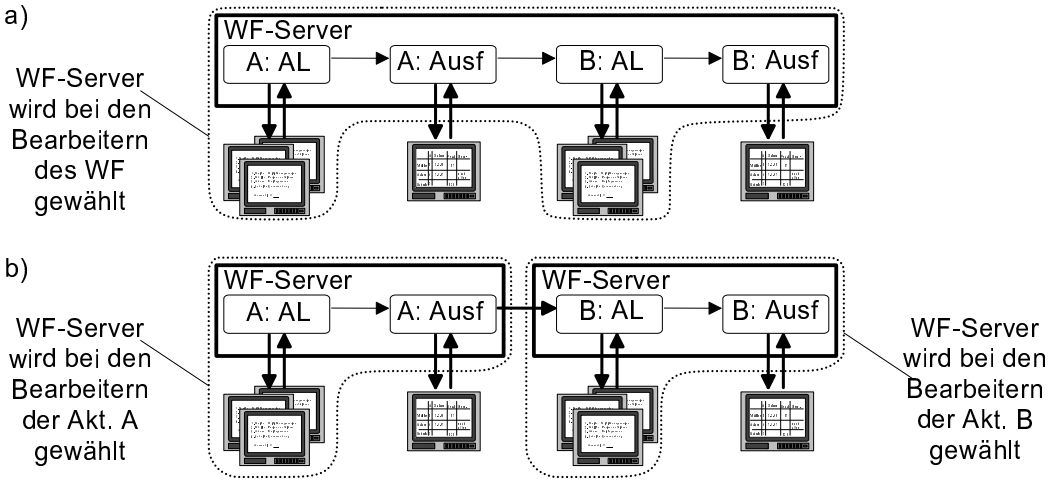


Abbildung 9.5 WF-Server nahe bei den potentiellen Bearbeitern
Granularität: a) gesamter Prozess (ohne Migration), b) einzelne Aktivitäten (mit Migration).

Ist eine Migration von WF-Instanzen möglich, so kann im Prinzip jede Aktivität von dem für sie optimalen Server kontrolliert werden. Allerdings gehen alle in der Literatur beschriebenen Ansätze dieser Kategorie von der einschränkenden Annahme aus, dass alle potentiellen Bearbeiter einer Aktivität derselben OE angehören. Die Bearbeiterauflösung erfolgt dann nur lokal für die Benutzer im Domain dieses WF-Servers. Es gibt also keine Ansätze, welche die Möglichkeit zur Migration mit einer globalen Bearbeiterauflösung kombinieren.

Wir analysieren zuerst die Variante ohne Migration. Die WF-Server sind jeweils für verschiedene WF-Typen zuständig. Da deren Laufzeitdaten voneinander unabhängig sind, ist keine Synchronisation zwischen den WF-Servern notwendig. Die Last für die Steuerung der Aktivitätenausführung wird unter den Servern aufgeteilt. Das Verteilen der Last ist etwas schwierig, da Prozesse nur als Ganzes

einem Server zugeordnet werden können. Sollte ein WF-Typ alleine schon einen Server überlasten, gibt es keine geeignete Verteilung.

Auch wenn Migrationen möglich sind, teilt sich die Last für die Steuerung der Aktivitätenausführung zwischen den WF-Servern auf. Es kommen zwar die Migrationskosten hinzu, aber die durch die Aktivitätenausführung erzeugte Kommunikationslast ist geringer, da durch die Migration für jede Aktivität ein geeigneter WF-Server ausgewählt werden kann.

Zur Abschätzung der Belastung der Server durch das Aktualisieren der Arbeitslisten nehmen wir generell an, dass ein Benutzer nicht in alle WF-Typen bzw. Aktivitätentypen involviert ist. Deshalb ist jeder Server nur für einen Teil der Benutzer zuständig, weshalb er auch nur einen Teil der Last bewältigen muss. Erfolgt die Bearbeiterauflösung nur lokal, d.h. ist der Server nur für die Bearbeiter seines Domains zuständig, so erzeugt das Aktualisieren der Arbeitslisten besonders wenig Last. Allerdings hat diese Variante den Nachteil, dass beim Ausfall eines Servers alle seine Benutzer blockiert sind, da sie ihre Aufträge nur von ihm erhalten. Außerdem sind manchmal Aktivitäten wünschenswert, die in mehreren OE bearbeitet werden können (z.B. Einholen einer Unterschrift von einem beliebigen Abteilungsleiter). Diese sind bei lokaler Bearbeiterauflösung nicht modellierbar. Des Weiteren schränkt dieses Konzept die Replikation der Server ein. Bei Überlastung eines Servers ist es nicht möglich, die Last auf mehrere Server zu verteilen, da diese dann getrennte OE mit disjunkten Benutzern darstellen würden.

Beispiele:
MOBILE [BHJ+96, HS96, Jab97] verfolgt das Ziel, durch Replikation der Server und Partitionierung der Daten die Last für die einzelnen Server zu minimieren und dadurch eine möglichst hohe Gesamtlast zu bewältigen. Dazu werden die Aufgaben eines WF-Servers in sogenannte Aspekte zerlegt, wie z.B. den funktionalen Aspekt, den Kontrollfluss und den Datenfluss. Jeder dieser Aspekte wird durch einen eigenen Server realisiert, die durch einen Systemkern verbunden sind. Ist ein solcher Server überlastet, so wird er repliziert.

Das Synchronisationsproblem bei Änderung der Daten dieser Server wurde folgendermaßen gelöst: Statische Daten, wie z.B. Schemadaten, werden repliziert. Die Server für verschiedene Aspekte haben keine gemeinsamen Daten, außer der statischen WF-ID; diese wird repliziert. Die anderen Daten lassen sich nach den Aspekten partitionieren. Die verschiedenen Replikate eines Servers für denselben Aspekt verwenden zwar dieselben dynamischen Instanzdaten, aber jeder WF-Typ hat eine korrespondierende OE und damit einen fest zugeordneten Server. Es werden also gesamte Prozesse einem WF-Server zugeordnet, eine Migration ist nicht vorgesehen. Entsprechend dieser Zuordnung lassen sich diese Daten partitionieren.

Die Aufteilung in Aspekte bringt eine Reduktion der Last um bestenfalls einen Faktor, welcher der Anzahl der verschiedenen Aspekte entspricht. Da dies aber ein kleiner (konstanter) Faktor ist (gemäß [HS96]: 6), führt dies zu keiner signifikanten Reduktion der Last. Dass sich die Last gleichmäßig auf die Aspekte-Server verteilt, ist zudem unrealistisch, da deren Aufgaben völlig unterschiedlichen Aufwand erfordern. Hinzu kommt, dass für bestimmte Operationen (wie z.B. die dynamische Restrukturierung eines Ablaufgraphen) fast alle Aspekte benötigt werden, so dass durch die Verteilung auf mehrere Server ein zusätzlicher Kommunikations- und Synchronisationsaufwand entsteht.

In [NSS98, Sch97b, SNS99] wurde der MOBILE-Ansatz noch folgendermaßen erweitert: Beim Start eines (Sub-)WF, also zur Ausführungszeit, wird entschieden, von welchem WF-Server dieser kontrolliert werden soll. Bei dieser Entscheidung werden Rechte, Gewichte (gewünschte Ressourcenzuordnung) und Kosten (Leistungsfähigkeit von Rechnern und Kommunikationsverbindungen) berück-

sichtigt. Es findet allerdings keine Migration der WF-Instanz statt, sondern die Subprozesse werden entfernt gestartet. Diese Vorgehensweise führt zu den in Abschnitt 3.4.2 diskutierten Nachteilen

Das **MENTOR**-System [WWWK96a, WWWK96b, WWK+97, MWW+98] basiert auf einigen Standardkomponenten, wie dem Transaktionsmonitor TUXEDO [UNI92], CORBA [OMG95a, OMG95b, OMG95c] und Statemate, einem Werkzeug zur Modellierung und Ausführung von State-/Activitycharts. Dabei wird in einem Statechart der Kontrollfluss und in einem Activitychart der zugehörige Datenfluss und der funktionale Aspekt eines WF modelliert. Kernidee des in diesem Projekt verfolgten Ansatzes ist es nun, die State-/Activitycharts so zu partitionieren, dass eine zum zentralen Fall äquivalente verteilte Ausführung entsteht und jede Aktivität in einem zuvor für sie nach OE festgelegten „Processing Entity“ ausgeführt wird. Der entsprechende Server ist somit nur für die ihm zugeordneten Aktivitäten und für die seinem Processing Entity angehörigen Benutzer zuständig. Dies ist einer der Ansätze, die auf eine globale Bearbeiterauflösung verzichten.

An den Zerlegungspunkten des State-/Activitycharts erfolgt ein Wechsel des Servers, d.h. die komplette WF-Instanz migriert zu einem anderen Server. Es werden nur globale Variablen verwendet, deren Änderungen durch einen in jedem Server vorhandenen Communication-Manager propagiert werden. Alle Kommunikationen werden durch ein 2PC geschützt, weshalb die verwendeten Anwendungen das XA-Protokoll unterstützen müssen.

WIDE [CGP+96, CGS97] verfolgt einen ähnlichen Ansatz, allerdings ist die Skalierbarkeit in diesem Projekt nur ein Teilaspekt. Es wird auch ein Transaktionsmanagement auf verschiedenen logischen Ebenen (geschachtelte Transaktionen für Aktivitäten bzw. Sagas für Prozesse) angeboten [GVBA97]. Mit Hilfe von aktiven Regeln ist zudem eine Ausnahmebehandlung möglich. Die Idee zur Verteilung ist analog zu MENTOR. Allerdings ist noch keine Migration vorgesehen, sondern die Daten verbleiben an einem Ort, und es wird entfernt mit Hilfe von CORBA-Diensten auf sie zugegriffen. Dies kann zu den schon erwähnten hohen Kosten bei mehrfachem weit entfernten Zugriff führen.

Diskussion:
Auch ADEPT fällt in die Kategorie, bei der der WF-Server nahe bei den potentiellen Bearbeitern der aktuellen Aktivität gewählt wird. ADEPT ist allerdings der einzige Ansatz, bei dem Migrationen mit einer globalen Bearbeiterauflösung kombiniert werden. Dies ist wichtig, um den WF-Server einer Aktivität ohne Beschränkungen festlegen zu können, so dass dadurch die Minimierung der Kommunikationskosten ermöglicht wird. Die durch dieses Verfahren gewonnenen Freiheitsgrade führen dazu, dass die Migrationskosten in ADEPT wesentlich niedriger als bei MENTOR und WIDE sind, da nicht bei jedem Wechsel der OE migriert werden muss.

9.1.3.3 Server sind nahe bei der Anwendung

Eine weitere Möglichkeit, um aus der Wahl des Ortes des WF-Servers Vorteile zu ziehen, ist ihn dort zu platzieren, wo das Anwendungsprogramm der entsprechenden Aktivität läuft (Abb. 9.6). Diese Variante wird vor allem von Systemen verwendet, die auf einer objektorientierten Infrastruktur (z.B. CORBA) basieren. Die Anwendung ist dabei als Objekt gekapselt, das an einem bestimmten Ort im verteilten System allokiert ist. Das entsprechende Teilnetz oder sogar derselbe Rechner wird auch für den WF-Server der zugehörigen Aktivität gewählt. Die WF-Instanz ist ein Objekt, das zu dem jeweiligen WF-Server migriert. Eine andere Möglichkeit ist, dass lediglich Objektreferenzen zwischen den Servern wandern und diese dann bei der Verwendung der zugehörigen Parameter entfernt auf die entsprechenden Objekte zugreifen.

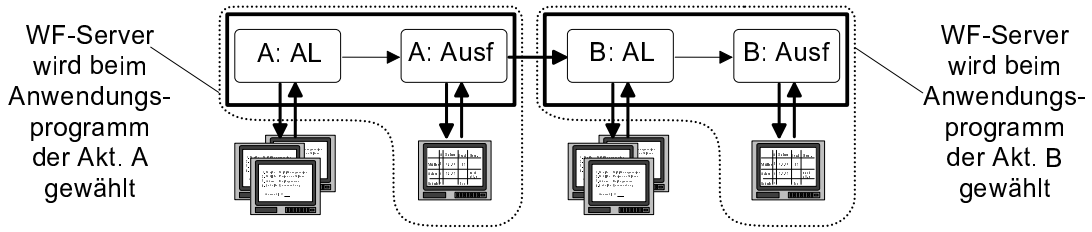


Abbildung 9.6 Der WF-Server wird nahe bei Anwendung platziert.

Die durch die Aktivitätenausführung erzeugte Last lässt sich bei diesem Konzept auf die einzelnen Server verteilen. Die dabei entstehende Migrationslast ist allerdings recht groß, da das Instanzenobjekt bei fast jedem Teilschritt migriert (selbst wenn er vom selben Bearbeiter wie der Vorgängerschritt ausgeführt wird), weil gewöhnlich eine andere Anwendung verwendet wird. Falls vom WF-Server entfernt auf die Daten zugegriffen wird, verringert dies zwar die Migrationslast, aber die Last durch die Aktivitätenausführung ist wesentlich größer, da die Anwendungen mehrfach auf ein (evtl. weit) entferntes Objekt zugreifen. Ein generelles Problem des Ansatzes ist, dass die Verteilung der Last davon abhängt, wie häufig die einzelnen Anwendungen aufgerufen werden. Dies lässt sich nicht immer steuern.

Es ist schwierig, die Belastung der Server durch das Aktualisieren der Arbeitslisten zu bestimmen. Ein Server muss prinzipiell alle Benutzer bedienen, da die Verteilung nicht an OE (z.B. Abteilungen) orientiert ist, sondern an Anwendungen. Dadurch kann er zu einem Flaschenhals werden. Allerdings werden i.d.R. die meisten Benutzer nur wenige Anwendungen aufrufen dürfen, da sie ihre Rolle auf bestimmte Tätigkeiten einschränkt. Da deshalb jede Anwendung nur von bestimmten Benutzern verwendet wird, müssen nur deren Arbeitslisten gewartet werden, was die Last für den WF-Server reduziert. Diese Annahme muss aber nicht gelten. Es sind Szenarien denkbar, in denen es Abteilungen gibt, deren Mitarbeiter zwar getrennte Prozesse bearbeiten, jedoch mit denselben Anwendungen. In einem solchen Fall wird jede Anwendung von jedem Benutzer aufgerufen, wodurch die Skalierbarkeit eingeschränkt wird.

Dieser Ansatz geht zudem von der Annahme aus, dass jede Anwendung einen eindeutig bestimmbaren Ort hat. Das ist auch der Grund dafür, dass dieses Verteilungsmodell vor allem von Ansätzen verwendet wird, die auf einer objektorientierten Infrastruktur basieren. Da das Aktivitätenprogramm dann als Objekt gekapselt ist, hat es stets einen bestimmbaren Ort. Bei einer Datenbankanwendung wäre dieser z.B. durch den Rechner bestimmt, auf dem das DBS läuft. Problematisch sind Anwendungen, die auf mehreren Rechnern installiert sind oder über das Netzwerk gestartet werden, wie z.B. ein Editor, eine Textverarbeitung oder eine Eingabemaske. Solche Anwendungen werden immer auf dem Rechner des (a priori unbekannten) Benutzers ausgeführt, der die Aktivität bearbeitet. Da durch die Anwendung kein Ort vorgegeben wird, ist dieser Ansatz nicht ohne weiteres verwendbar, weil dann kein WF-Server für die entsprechende Aktivität bestimmt werden kann. Man könnte nun für den WF-Server einen beliebigen Ort wählen, dies wäre aber bezüglich der Kommunikationskosten sehr ungünstig. Es bietet sich an, den WF-Server in diesen Fällen wie bei den Ansätzen aus Abschnitt 9.1.3.2 so zu wählen, dass er nahe bei den Bearbeitern der Aktivität liegt. – In der Regel werden Anwendungen wohl häufig auf Rechnern der OE laufen, zu der ihre Bearbeiter gehören, so dass die letzten beiden Verteilungsmodelle zu ähnlichen Ergebnissen führen.

Beispiele:
Die verwandten Systeme **CodAlf** [SM96, SM97] und **BPAFrame** [HIM96, Mit97, MIZ+96, MMSL96, SM96] basieren auf unterschiedlicher objektorientierter Middleware. Während CodAlf

eine objektorientierte Erweiterung des OSF DCE [Sch93] verwendet, basiert BPAFrame auf CORBA [OMG95a, OMG95b, OMG95c]. Wie oben beschrieben, wird jede Anwendung in einem Objekt gekapselt. Ein solches Objekt hat einen festen Ort – an dem sich auch der zugehörige WF-Server befindet – und wird als Runtime-System bezeichnet. WF-Typen werden als Objekttypen implementiert, die WF-Instanzen sind somit als Objekte realisiert. Sie enthalten außer den Anwendungsdaten auch noch die Prozessbeschreibung. Diese mobilen Objekte migrieren zum Ort des Anwendungsobjektes der nächsten Aktivität. Dieser Ort wird von einer Komponente (Trader) ermittelt, die als Directory Service dient (also angibt, wo sich ein geeignetes Objekt befindet) und außerdem eine Lastverteilung vornimmt. Der erforderliche entfernte Zugriff auf Daten wird durch die verwendeten Middleware-Dienste realisiert. Dasselbe gilt für den Zugriff der Benutzer auf die entfernten Runtime-Systeme.

Wie oben bereits ausgeführt wurde, hängt bei diesem Ansatz die Last eines WF-Servers davon ab, wie häufig die zugehörige Anwendung verwendet wird. Falls es aber möglich ist, den Anwendungsserver zu replizieren, so sorgt der Trader für eine Verteilung der Last.

METEOR$_2$ [DKM$^+$97, MPS$^+$98, MSKW96, SK97] verwendet einen Ansatz, der dem eben beschriebenen sehr ähnlich ist. Er basiert ebenfalls auf CORBA. Allerdings ist die Ablaufbeschreibung nicht in einem mobilen Instanzenobjekt enthalten, sondern sie wird in ihre Teilschritte (jeweils mit Kanten zu vorangehenden und nachfolgenden Aktivitäten) zerlegt. Aus dieser Information wird für jede Aktivität ein Teil des WF-Servers (Taskmanager genannt) erzeugt, der die Ausführung genau dieser Aktivität steuert. Ein Taskmanager kann prinzipiell an einem beliebigen Ort allokiert werden, allerdings wird auch hier der Ort der Anwendung vorgeschlagen. Im Gegensatz zu den beiden vorherigen Ansätzen migrieren keine Instanzenobjekte zwischen den Taskmanagern. Stattdessen werden beim Weiterschalten des Prozesses zur nächsten Aktivität Referenzen auf die verwendeten Datenobjekte übergeben. Bei der Ausführung einer Aktivität muss dann auf die i.d.R. entfernt liegenden Datenobjekte zugegriffen werden. Schließlich wird in METEOR$_2$ der Ort der Anwendungen und der Taskmanager eindeutig festgelegt, so dass die Aufgabe des Traders entfällt. Dadurch ist aber auch keine dynamische Lastbalancierung möglich.

Der Ansatz betrachtet auch noch weitere Aspekte, wie den Wiederanlauf nach Fehlern oder die effiziente Ermittlung des aktuellen Zustands einer WF-Instanz. Zu diesem Zweck gibt es einen zentralen Monitoring-Service, dem die Taskmanager Änderungen der Zustände melden. Außerdem werden ihm Referenzen auf verwendete Daten geschickt, um die Daten beim Wiederanlauf nach Fehlern rekonstruieren zu können.

Außer der Last der WF-Server ist bei diesem Ansatz auch noch die Belastung des Monitoring-Services interessant. Da diese zentrale Komponente bei der Ausführung jeder Aktivität über Änderungen informiert werden muss, ist ihre Last proportional zu der Last, die durch die Aktivitätenausführung erzeugt wird. Sie ist allerdings nur ein sehr kleiner Bruchteil dieser Last, da lediglich Referenzen auf Daten übertragen und nur einfache Lese- und Schreiboperationen durchgeführt werden. Aus diesem Grund ist eine zentrale Komponente bis zu einer sehr hohen Systemlast ausreichend. Etwas schlechter sieht die Analyse für das Teilnetz aus, in dem der Monitoring-Service angesiedelt ist. Es werden zwar nur kleine Pakete übertragen, dafür aber sehr viele. Der Monitoring-Service stellt auch ein Problem für die Verfügbarkeit dar. Da er für das Recovery notwendig ist, führt sein Ausfall zum Systemstillstand. Eine Replikation ist nicht vorgesehen, sie würde im Fall einer Netzpartitionierung auch nicht verhindern, dass der abgeschnittene Teil (u.U. ist das der größere Teil) blockiert ist.

Diskussion:
Beim Aktualisieren von Arbeitslisten und beim Starten von Aktivitätenprogrammen erfordert dieses Verteilungsmodell mindestens so hohe Kommunikationskosten wie der in Abschnitt 9.1.3.2 vorge-

stellte Ansatz, da die WF-Server potentiell weiter von den von diesen Aktionen betroffenen Benutzern entfernt sind. Da sich die Verteilung an der Lage der Anwendungsprogramme und nicht an deren Bearbeitern orientiert, können bei diesem Ansatz keine variablen Serverzuordnungen eingesetzt werden. Dies führt dazu, dass das günstige Kommunikationsverhalten von ADEPT bei diesem Verteilungsmodell nicht erreicht wird.

9.1.4 Voll verteilte Workflow-Management-Systeme

Ein WF-Server und das zugehörige Teilnetz sind potentielle Engpässe eines Systems. Diese können vermieden werden, indem man auf (explizite) Server verzichtet und die entsprechende Funktionalität in jedem Client realisiert. Die komplette WF-Instanz migriert dann nach Beendigung einer Aktivität und der Ermittlung des nächsten Bearbeiters zu dessen Rechner (siehe Abb. 9.7). Das heißt, jeder Arbeitsplatzrechner fungiert quasi als „Mini-Server“ für seine lokalen Anwendungen. Beim Ermitteln dieses Bearbeiters wird eine verteilte Synchronisation notwendig. Hierbei stellt sich die Frage, woher ein Client alle angemeldeten Benutzer mit einer passenden Rolle für die Nachfolgeraktivität kennt. Dies ist notwendig, um einen entsprechenden Eintrag in deren Arbeitslisten einfügen zu können. Dieses Problem wird von manchen Systemen dadurch umgangen, dass man auf die Bearbeiterauflösung verzichtet und jeder Aktivität einen eindeutigen Bearbeiter zuordnet.

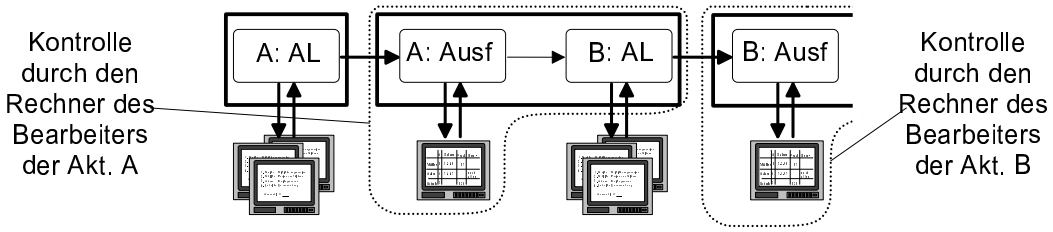


Abbildung 9.7 WF-Kontrolle durch den Rechner des Benutzers, der die aktuelle Aktivität ausführt.

Die durch die WF-Ausführung entstehende Last verteilt sich auf alle Arbeitsstationen des Systems. Hierdurch ist die Last pro Komponente („Mini-Server“) kleiner als bei den in Abschnitt 9.1.3 beschriebenen Systemen. Das ist auch notwendig, da die Arbeitsstationen keine leistungsstarken Maschinen, sondern nur die Rechner der Benutzer sind. Die Migrationslast ist bei voll verteilten Ansätzen besonders hoch, weil die WF-Instanz bei fast jedem Teilschritt migriert. Der dafür erforderliche Aufwand teilt sich ebenfalls auf die Arbeitsstationen aller Benutzer auf. Wird auf eine Bearbeiterauflösung verzichtet, so wird der Aufwand für die verteilte Synchronisation vermieden. Dies stellt aber einen Verlust an Funktionalität bzw. Flexibilität dar, der nicht bei jeder Anwendung akzeptiert werden kann. Wird eine Bearbeiterauflösung durchgeführt, so verteilt sich die dadurch entstehende Last auf die vielen Arbeitsstationen der Benutzer, weshalb die Last für eine einzelne Maschine klein ist. Da es keine WF-Server gibt, bei denen sich die Kommunikation konzentriert, gibt es keine Flaschenhälse im Kommunikationsnetzwerk. Die Belastung der Teilnetze stellt also kein Problem dar, wenn die Arbeitsstationen auf ausreichend viele Teilnetze verteilt werden. Problematisch ist, dass die Zustandsinformation nicht in Servern verfügbar, sondern über die Benutzerrechner verteilt ist. Deshalb muss der WF „gesucht“ werden, wenn nach seinem aktuellen Zustand gefragt wird, was zu einem hohen Aufwand führt.

Beispiele:
INCAS [BMR94, BMR96] ist ein System, das auf eine Bearbeiterauflösung verzichtet. Der Name kommt von einem INformation CArrier, der jeweils zur Arbeitsstation der nächsten Aktivität migriert. Dieser INCA enthält den gewünschten Dienst, Regeln, die den Daten- und Kontrollfluss beschreiben,

die Daten der WF-Instanz und Atomaritätsanforderungen. Auch die Arbeitsstationen verfügen über Regeln, die mit denen des INCA zusammen verwendet werden, um eine Aktivität auszuführen und die nächste Aktivität zu berechnen. Für diese Nachfolgeraktivität wird, ebenfalls anhand der Regeln, die zugehörige Arbeitsstation berechnet. Es findet keine Bearbeiterauflösung statt, somit ist auch keine Synchronisation zwischen den potentiellen Nachfolgern notwendig. Bei jedem Weiterschalten zur nächsten Aktivität wird der INCA verändert. Er wird allerdings nicht modifiziert, sondern es wird eine neue Version von ihm erzeugt, die zusammen mit der alten migriert. Da dadurch die WF-Instanz sehr groß wird, ist die durch die Migration verursachte Last besonders hoch. Ein Vorteil dieser Vorgehensweise ist jedoch, dass sich Regeln auch auf alte Versionen der Daten beziehen können. Das gesamte System wird durch Regeln gesteuert. Die Regelmenge ist sogar dynamisch, d.h. bei der Ausführung einer Aktivität können Regeln erzeugt oder verändert werden. Auch die Transaktionssemantik der Aktivitäten wird mittels Regeln definiert. Allerdings ist eine große, verteilte und dazu noch dynamische Regelmenge schwer zu durchschauen.

Bei **Exotica/FMQM** [AMG+95] wird nach Beendigung einer Aktivität eine globale Bearbeiterauflösung durchgeführt. Die Kommunikation findet (gesichert) über Persistent-Queues statt. Der Ablauf wird wie in FlowMark durch einen Graphen mit Kontroll- und Datenkonnektoren beschrieben. Dieser Graph wird so auf die Knoten verteilt, dass jeder Knoten diejenigen Teile des Graphen erhält, die Aktivitäten mit den Rollen seiner Benutzer enthalten. Ist die Bearbeitung einer Aktivität beendet, so werden Nachrichten an alle Knoten geschickt, die für Aktivitäten verantwortlich sind, zu denen von der aktuellen Aktivität aus Kontroll- oder Datenkanten führen. Die Daten werden bei dieser Methode schrittweise verteilt, sie migrieren nicht mit der WF-Instanz.

Wegen der Bearbeiterauflösung kann eine Instanz nicht einfach an den Knoten der nachfolgenden Aktivität gesendet werden, sondern dieser muss erst ermittelt werden. Dazu informiert der Vorgängerknoten alle potentiellen Nachfolger über die zu vergebende Aktivität, indem er die entsprechende Information in deren Message-Queues schreibt. Diese holen sich dann die Instanz aus dessen Ausgabe-Queue, wenn sie die nächste Aktivität ausführen wollen. Die Synchronisation erfolgt durch das Transaktionskonzept des Queuing-Systems. Durch die Auswertung der von einer Aktivität ausgehenden Datenkonnektoren lassen sich aber lediglich die Aktivitäten ermitteln, die diese Daten potentiell benötigen. Es ist jedoch noch nicht bekannt, an welchen Knoten diese später ausgeführt werden, so dass die Instanzdaten zu ihnen transportiert werden können. Die bei diesem Ansatz erzeugte Last hängt stark davon ab, wie effizient der persistente Message-Queue-Dienst implementiert ist, insbesondere ob das entfernte Lesen (effizient) unterstützt wird.

Diskussion:
Auch bei diesem Ansatz findet die Kontrolle einer Aktivitäteninstanz nahe beim ausführenden Benutzer statt. Dadurch wird sogar ein Verhalten erreicht, das dem bei Verwendung variabler Serverzuordnungen ähnelt, da eine Aktivität bei verschiedenen WF-Instanzen von unterschiedlichen – jeweils geeigneten – Servern kontrolliert werden kann. Allerdings führt dieses Verteilungsmodell, wegen der zahlreichen erzwungenen Migrationen und der aufwendigen Bearbeiterauflösung, zu sehr viel Kommunikation. Deshalb ist das Verteilungsmodell von ADEPT in unternehmensweiten Anwendungen vorzuziehen.

9.1.5 Sonstige Ansätze

In der Literatur finden sich auch einige Ansätze für verteiltes WF-Management, die keine Aussagen zum verwendeten Verteilungsmodell machen, d.h. es wird nicht angegeben, wie die zu erledigenden

Aufgaben auf die einzelnen Rechner des WfMS verteilt werden sollen. Im Folgenden werden einige diese Ansätze kurz vorgestellt.

Beispiele:
Im **METUFlow**-Projekt [DGA+97] werden außer Verteilungsaspekten [GAC+97] auch Transaktionskonzepte für Aktivitäten [AAHD97, KAK+98] und Strategien für Anfragen gegen eine verteilte WF-Historien-DB [KAD98] betrachtet. Um verteiltes WF-Management zu realisieren, wird ein blockbasiertes WF-Modell zu Regeln (Guards) übersetzt, durch welche die WF-Ausführung gesteuert wird. Für jede Aktivitäteninstanz wird ein Guard-Handler erzeugt, der für die Kommunikation mit den Guard-Handlern anderer Aktivitäten zuständig ist und die für diese Aktivität relevanten Guards evaluiert. Anhand der Werte dieser Guards wird entschieden, ob eine bestimmte Aktion durchgeführt werden kann (z.B. sind alle Vorbedingungen für den Start dieser Aktivität erfüllt, d.h. sind alle Vorgängeraktivitäten schon beendet). Allerdings werden bei diesem Ansatz keine Aussagen darüber gemacht, nach welcher Strategie die Guards auf die zur Verfügung stehenden Rechner verteilt werden, so dass nicht auf das verwendete Verteilungsmodell geschlossen werden kann.

WASA$_2$ [WHKS98] kümmert sich primär um dynamische Abweichungen vom WF-Schema [Wes98, Wes99a]. Allerdings wird in diesem Zusammenhang auch die verteilte WF-Ausführung betrachtet [Wes99b, Wes99c]. Um eine solche zu erreichen, werden die Aktivitäten als CORBA-Objekte realisiert, die sich Zustandsänderungen signalisieren. Da diese Objekte auf beliebigen Rechnern platziert werden können, lässt sich so auf einfache Weise ein verteiltes WF-Management realisieren. Allerdings wird auch in diesen Arbeiten keine Aussage über die Strategie bei der Verteilung der CORBA-Objekte gemacht.

Auch das **MOKASSIN**-Projekt beschäftigt sich primär mit der Adaption von WF-Instanzen [JH98]. Die Verteilung [GJS+99, JH99, Joe99] ist ähnlich wie beim vorherigen Ansatz realisiert: Die Aktivitäteninstanzen stellen CORBA-Objekte dar, die durch Events miteinander kommunizieren und auf beliebigen Rechnern platziert werden können. Auch bei diesem Ansatz wird keine Aussage über das Verteilungsmodell gemacht.

Diskussion:
Um die Belastung eines WF-Servers zu reduzieren, ist jede beliebige Verteilung der Aufgaben auf mehrere Rechner geeignet. Die soeben vorgestellten Ansätze ermöglichen eine beliebige (im Sinne von: zufällige) Verteilung der Aufgaben. Diese führt aber zu einem äußerst ungünstigen Kommunikationsverhalten, so dass die in ADEPT$_{distribution}$ anvisierten Ziele mit diesen Ansätzen nicht erreicht werden können.

9.1.6 Auswahl eines geeigneten Verteilungsmodells

Dieser Abschnitt fasst die in dem Abschnitt bisher gewonnenen Ergebnisse zusammen und bietet so einen Überblick über die vorgestellten Verteilungsmodelle. Außerdem wird skizziert, wie ein geeignetes Verteilungsmodell für eine konkrete Anwendung ausgewählt werden kann.

Es wurden drei fundamentale Klassen von Verteilungsmodellen für WfMS identifiziert. Dies sind zum einen Systeme mit einem zentralen Server. Da dieser einen potentiellen Engpass darstellt, muss bei einer hohen Benutzerzahl ein entsprechend leistungsstarker und damit teurer Rechner verwendet werden. Die dafür aktuell zur Verfügung stehende Technologie bildet eine Obergrenze für die Leistungsfähigkeit des WfMS. Weitere Klassen bilden Systeme mit mehreren Servern und Systeme ohne Server, bei denen die Ablaufsteuerung vollständig verteilt ist. Innerhalb dieser Klassen wurden

noch Unterklassifikationen vorgenommen. Einige kommerzielle Systeme und Vorschläge aus dem Bereich der Forschung wurden bezüglich dieser Klassifikation eingeordnet und auf ihre Skalierbarkeit hin untersucht. Tabelle 9.1 bietet eine Übersicht über die wichtigsten Eigenschaften der diskutierten Verteilungsmodelle.

Modell	Eigenschaften
WfMS mit zentralem Server:	
	der WF-Server und dessen Teilnetz bilden potentielle Flaschenhälse
	es gibt wenig Möglichkeiten zur Optimierung der Lage des WF-Servers
WfMS mit mehreren Servern	
Server werden zufällig gewählt:	
	ist einfach zu realisieren, kein Zusatzaufwand durch Migrationen
	keine Optimierung bzgl. Kommunikationskosten möglich
	jeder Cluster (WF-DB) muss Arbeitslisten aller Benutzer verwalten
Server sind nahe bei den Bearbeitern:	
	Last verteilt sich auf Server, gute Verteilung der Kommunikationslast möglich
	ohne Migration: Server ist bzgl. einzelner Aktivitäten nicht optimal
	lokale Bearbeiterauflösung: eingeschränkte Funktionalität
	variable Serverzuordnungen: weitere Reduzierung der Kommunikationslast
Server sind nahe bei der Anwendung:	
	erfordert, dass Aktivitätenprogramme einen fest zugeordneten Ort haben
	meist in Kombination mit objektorientierter Infrastruktur verwendet
	ist häufig ähnlich wie das Verteilungsmodell „Server bei den Bearbeitern“
voll verteilte WfMS:	
	keine Flaschenhälse im System vorhanden
	viele Migrationen (bei jedem Wechsel des Bearbeiters)
	problematisch: sehr hoher Grad an Verteilung der Information
	häufig muss jede Aktivität einen fest zugeordneten Bearbeiter haben

Tabelle 9.1: Die wichtigsten Eigenschaften der untersuchten Verteilungsmodelle.

Um ein geeignetes Verteilungsmodell für ein WfMS in einem konkreten Anwendungsfall auswählen zu können, ist es notwendig, die jeweilige Anwendung genau zu analysieren. Ggf. sollten die verschiedenen Verteilungsmodelle dazu durch eine Simulation dieser Anwendung verglichen werden. Generell ist ein zentrales WfMS nur für Anwendungen geeignet, bei denen die Anzahl der Benutzer beschränkt ist. Voll verteilte Ansätze führen wegen ihrem hohen Grad an Verteilung der Information zu Nachteilen. Sie eignen sich eigentlich nur, wenn fast jeder Aktivität eindeutig ein Bearbeiter zugeordnet ist. Haben Sequenzen von Aktivitäten denselben Bearbeiter, so ergibt sich ein ähnliches Verhalten wie bei variablen Serverzuordnungen: der WF migriert einmal zum Rechner dieses Bearbeiters (in der entsprechenden OE) und verbleibt dort für die Ausführung mehrerer Aktivitäten. Da sich der WF sogar auf dem richtigen Rechner (nicht nur in der richtigen OE) befindet, entfällt dann die Kommunikation für den Transfer der Parameterdaten des Aktivitätenprogramms. Es bleibt zu bemerken, dass diese beiden Verteilungsmodelle nur in Ausnahmefällen für ganz spezielle Anwendungen geeignet sind.

Für unternehmensweite Anwendungen ist meist nur ein Verteilungsmodell mit mehreren Servern geeignet. Bei diesem kann die Last auf eine überschaubare und geeignet gewählte Anzahl von WF-Servern verteilt werden. Es empfiehlt sich normalerweise nicht, den WF-Server zufällig zu wählen,

da dann kein Vorteil aus dessen Wahl gezogen werden kann. Den Server nahe bei der Anwendung zu platzieren ist schwierig, da für Anwendungen häufig kein Ort vorgegeben ist. Außerdem kann diese Strategie zu schlechten Antwortzeiten führen, falls der WF-Server durch diese Festlegung weit von den Clients entfernt liegt. Deshalb ist es für das Kommunikationsverhalten i.d.R. am günstigsten, den Server nahe bei den Bearbeitern der Aktivitäten zu wählen. Ob mittels variabler Serverzuordnungen weitere Verbesserungen erreicht werden können, hängt von der betrachteten Anwendung ab. Dies ist immer dann der Fall, wenn abhängige Bearbeiterzuordnungen verwendet werden. Durch eine Einschränkung der Funktionalität eines WfMS ist ebenfalls eine Verringerung der Kommunikationskosten möglich. Werden z.B. in der zu realisierenden Anwendung die Prozesse fast komplett innerhalb nur einer OE ausgeführt, so kann auf Migrationen verzichtet werden. Gehören alle Bearbeiter einer Aktivität jeweils zur selben OE, so ist keine globale Bearbeiterauflösung notwendig. Die maximal mögliche Funktionalität eines WfMS und das maximale Optimierungspotential bei der Festlegung der Verteilung erhält man aber nur, wenn Migrationen und globale Bearbeiterauflösung kombiniert werden können. Deshalb wurde für ADEPT ein Verteilungsmodell ausgewählt, das dies ermöglicht.

9.2 Simulationsumgebung

Der verbleibende Teil dieses Kapitels behandelt die durchgeführten Simulationen. Im vorliegenden Abschnitt betrachten wir die verwendete Simulationsumgebung. Dazu werden die Anforderungen und die Ziele, die sie erfüllen soll, beschrieben. Anschließend werden einige der beim Entwurf des Simulationsprogramms getroffenen Entscheidungen diskutiert. Schließlich wird noch der Ablauf einer Simulation beschrieben, d.h. es wird erläutert, wie das Simulationsprogramm arbeitet. Wie die dabei entstehenden Ergebnisdaten ausgewertet werden, wird in Abschnitt 9.3 erläutert.

9.2.1 Anforderungen und Ziele

Eine Simulationsumgebung muss die folgenden Anforderungen erfüllen:

- Es muss möglich sein, beliebige Szenarien (d.h. Abläufe, Verteilungsmodelle, Systemkonfigurationen, Verteilung der Bearbeiter, u.s.w.) zu simulieren. Das Szenario soll nicht durch das Simulationsprogramm beschrieben werden, sondern konfigurierbar sein. Idealerweise wird es durch eine wiederverwendbare Eingabedatei beschrieben, so dass die durch eine Änderungen des Szenarios resultierenden Auswirkungen mit geringem Aufwand untersucht werden können.
- Das Simulationsprogramm soll die Last berechnen, die in einem realen WfMS bei der Ausführung der im Szenario beschriebenen WF-Instanzen entsteht. Die Last muss für alle Einzelkomponenten (WF-Server, Teilnetze, Gateways) des WfMS aufgeschlüsselt sein. Für jede dieser Einzelkomponenten sollen die Anzahl der pro Zeiteinheit ausgeführten Aktionen und das umgesetzte Datenvolumen berechnet werden. Die Aktionen müssen zusätzlich nach Typ (z.B. Aktivität starten, Arbeitsliste aktualisieren) aufgeschlüsselt sein.
- Um die Qualität der Simulationsergebnisse beurteilen zu können, muss es möglich sein, den darin enthaltenen Fehler abzuschätzen.

Eine Simulation ermöglicht es, die in einem bestimmten Szenario auftretende Last abzuschätzen. Dies ist aber kein Selbstzweck, sondern dient den folgenden Zielen:

- Durch die Simulationen soll quantitativ verifiziert werden, dass das Verteilungsmodell von ADEPT dazu geeignet ist, die Kommunikationslast signifikant zu reduzieren. Dies bezieht sich

sowohl auf die von ADEPT verwendete (statische) Partitionierung, als auch auf die Verwendung von variablen Serverzuordnungen.

- Der im Abschnitt 9.1 durchgeführte qualitative Vergleich der verschiedenen Verteilungsmodelle soll durch eine Abschätzung der in verschiedenen Szenarien auftretenden Last vervollständigt werden.
- Eine Simulation eines konkreten Szenarios ermöglicht es, die in diesem Szenario auftretende Last vorherzuberechnen. Dadurch ist es schon vor Einführung eines WfMS möglich, zu überprüfen, ob die Systemkomponenten überlastet sein werden.

Eine alternative Möglichkeit zum quantitativen Vergleich der Verteilungsmodelle ist die Abschätzung der entstehenden Belastung der WF-Server und des Kommunikationssystems durch Formeln. Dieser Ansatz wurde in [BD98] verfolgt. Dabei wurde z.B. für jedes Verteilungsmodell angegeben, die groß die durchschnittliche Last pro Komponente bei der Aktivitätenausführung bzw. bei der Arbeitslistenaktualisierung ist. Nachteile dieser Vorgehensweise sind, dass die resultierende Last nur sehr grob abgeschätzt werden kann, da es kaum möglich ist, alle Details eines Verteilungsmodells zu berücksichtigen. Außerdem ist es nicht möglich, die in einem bestimmten Szenario entstehende Last einer Einzelkomponente vorherzusagen, da die Formeln eher das Lastverhalten im Sinne der O-Notation [CLR96] beschreiben. Aus diesen Gründen verspricht die Verwendung von Simulationen wesentlich mehr Erfolg.

9.2.2 Entwurfsentscheidungen

Eine Simulation kann prinzipiell auf Basis eines Simulationspaketes, einer Simulationssprache [Hof98, Lan92] oder direkt auf Basis einer Programmiersprache entwickelt werden. Im Folgenden werden diese Alternativen untersucht und verglichen und eine geeignete ausgewählt.

Simulationspakete sind für Simulationen in einem vorgegebenen Anwendungsbereich geeignet. So bietet SIMULINK [Hof98] Elemente zur Simulation von dynamischen Systemen aus dem Ingenieurbereich (z.B. elektronische Schaltelemente). GPPS-FORTRAN [Lan92] dient zur Simulation von Abläufen innerhalb von Hardware-Komponenten. Zu diesem Zweck können Elemente wie Bedienstationen und Warteschlangen verknüpft und ihr Verhalten simuliert werden. Wegen der speziellen Funktionalität solcher Simulationspakete, sind sie zur Darstellung der komplexen Abläufe in einem WfMS kaum geeignet. Deshalb ist ihre Verwendung nicht sinnvoll.

Simulationssprachen wie MATLAB [Hof98] und SIMULA [Lan92] stellen i.d.R. eine Erweiterung einer Programmiersprache durch Makros und Bibliotheksfunktionen dar. Diese realisieren einfache, bei Simulationen häufig benötigte Funktionen, z.B. zur Verwaltung von Vektoren, Matrizen und Ereignislisten oder Funktionen zur statistischen Auswertung von Simulationsergebnissen (z.B. Berechnung von Mittelwerten und Konfidenzintervallen). Diese Funktionalität wird auch für die vorliegende Simulation benötigt, so dass die Verwendung einer Simulationssprache prinzipiell möglich ist.

Ein Simulationsprogramm kann auch direkt auf Basis einer Programmiersprache realisiert werden. Diese Alternative wurde im vorliegenden Fall gewählt, wobei die Programmiersprache C [KR90b] verwendet wurde. Gegen die Verwendung einer Simulationssprache spricht, dass dies die Ausführungszeit des Simulationsprogramms erhöht [Hof98, Lan92]. Andererseits sind in C geschriebene Programme sehr schnell. Dies gilt natürlich im Vergleich zu Programmen, die auf einer Simulationssprache basieren, aber auch im Vergleich zu in anderen Programmiersprachen implementierten Programmen, da der Befehlssatz von C sehr „maschinennah“ ist. Auf die Verwendung einer Simulationssprache wurde außerdem verzichtet, da die benötigten Datenstrukturen und mathematischen Funk-

tionen leicht selbst zu realisieren sind. Der hierfür notwendige Aufwand ist im Vergleich zur Realisierung der komplexen Ablaufsteuerung eines WfMS vernachlässigbar klein. Um die gewünschte Genauigkeit der Simulationsergebnisse (siehe Abschnitt 9.3) zu erhalten, ist die Simulation der Ausführung extrem vieler WF-Instanzen notwendig. Deshalb ergeben sich für einige der vorgestellten Simulationen Ausführungszeiten von bis zu 12 Tagen auf einer Sun ultra 10. Eine deutliche Erhöhung dieser Zeiten wäre nicht akzeptabel. Durch diese Ausführungszeiten wurde nachträglich bestätigt, dass die Entwurfsentscheidung richtig war, das Simulationsprogramm direkt in C zu implementieren. Im nachfolgenden Abschnitt wird die Funktionsweise des Simulationsprogramms kurz beschrieben.

9.2.3 Ablauf einer Simulation

Um einen Überblick über das Simulationsprogramm zu geben, wird zuerst die von ihm realisierte Funktionalität beschrieben. Anschließend wollen wir noch kurz seine interne Funktionsweise betrachten.

9.2.3.1 Sicht des Benutzers

Das Simulationsprogramm liest ein WF-Szenario ein und simuliert die Ausführung der WF-Instanzen durch die Benutzer des WfMS. Jede dabei ausgeführte Aktion und Kommunikation, sowie die zugehörige(n) Komponente(n) und der Zeitpunkt werden vermerkt, so dass die Belastung jedes Servers, Teilnetzes und Gateways berechnet werden kann. Die Last nach Beendigung der Einschwingphase (siehe Abschnitt 9.3.2) kann dann z.B. zum Vergleich der Verteilungsmodelle verwendet werden.

Im WF-Szenario wird das gesamte zu simulierende Modell festgelegt. Für eine detaillierte Auflistung aller vorzugebenden Parameter siehe Anhang C.1. Diese beinhalten z.B. allgemeine Daten wie die Simulationsdauer oder die Menge der beim Transport einer Arbeitsliste anfallenden Daten (abhängig von der Anzahl der Einträge). Für jeden Benutzer wird das Teilnetz, in dem er angesiedelt ist, seine Rolle und seine OE festgelegt. Für jeden WF-Typ wird angegeben, wie viele Instanzen simuliert werden sollen. Außerdem wird der Kontrollfluss vorgegeben. Für Aktivitäten wird die Größe der Ein- und Ausgabeparameter und die Bearbeiterzuordnung definiert. Letztere kann durch die Rolle der Benutzer erfolgen. Es können aber auch abhängige Bearbeiterzuordnungen festgelegt werden, indem der Bearbeiter einer vorangegangenen Aktivität oder dessen OE referenziert wird. Außerdem kann auch der Benutzer referenziert werden, der die WF-Instanz gestartet hat. Die Abbildung auf die verschiedenen Verteilungsmodelle erfolgt durch eine entsprechende Festlegung der Serverzuordnungen der einzelnen Aktivitäten. Dabei sind die folgenden Varianten möglich:

- Statisch: Der Server der Aktivität wird fest vorgegeben.
- Zufällig: Bei der Ausführung einer Instanz der Aktivität wird zufällig ein Server ausgewählt.
- $Server(m)$: Es wird derselbe Server verwendet, wie für die vorhergehende Aktivität m.
- $Domain(Bearb(m))$: Es wird der Server desjenigen Domains gewählt, zu dem der Bearbeiter der Aktivität m gehört. Bei dieser Serverzuordnung ist es auch möglich, denjenigen Benutzer zu referenzieren, der die WF-Instanz gestartet hat.
- Voll verteilt: die komplette WF-Instanz migriert auf den Rechner des Bearbeiters der aktuellen Aktivität.

Mit diesen Serverzuordnungen können alle im Abschnitt 9.1 beschriebenen Verteilungsmodelle nachgebildet werden. Wird z.B. der Server einer WF-Instanz zufällig ausgewählt (Abschnitt 9.1.3.1), so

wird der Server für die Startaktivität zufällig gewählt und dieser wird dann in den Serverzuordnungen aller weiteren Aktivitäten referenziert. Allerdings sind von den in ADEPT angebotenen Serverzuordnungen (vgl. Abschnitt 5.1.2) nur die Typen 1-3 realisiert. Variable Serverzuordnungen vom Typ 4-6 werden in den simulierten Beispielszenarien nicht benötigt. Dass es aber durchaus Fälle gibt, in denen auch diese Serverzuordnungsausdrücke benötigt werden, wurde schon in Abschnitt 5.1.3 gezeigt. Die Größenordnung der durch ihren Einsatz möglichen Reduktion der Kommunikationslast ist aber dieselbe, wie bei den realisierten Typen variabler Serverzuordnungen, da ihr Grundprinzip identisch ist. Deshalb hätten sich durch die Realisierung von Serverzuordnungen des Typs 4-6 keine neuen Erkenntnisse ergeben.

Bei der Entwicklung des Simulationsprogramms wurde darauf geachtet, dass sich die Simulation möglichst so verhält, wie die Bearbeitung in einem realen WfMS: Immer, wenn ein Bearbeiter eine Aktivität beendet hat, wählt er (zufällig) eine Aktivität aus seiner Arbeitsliste aus und startet sie. Die Startzeitpunkte der WF-Instanzen werden zufällig über den Simulationszeitraum verteilt, um nicht künstliche Lastspitzen für bestimmte Server oder Bearbeiter zu erzeugen. Die Arbeitslisten der Benutzer werden im Falle ihrer Änderung durch die WF-Server aktualisiert, wobei Mindestzeitabstände eingehalten werden (Verfahren aus Abschnitt 2.5.3). Deshalb findet keine Übertragung statt, wenn sich die Arbeitsliste nicht verändert hat, und es wird vermieden, dass bei häufiger Änderung ständig Kommunikationskosten anfallen. Für den Parametertransfer zu den Aktivitätenprogrammen werden die Datenmengen veranschlagt, die wirklich benötigt werden und bei Migrationen wird die aktuelle Größe der WF-Instanzdaten berücksichtigt. Dies entspricht dem in Abschnitt 2.3.1.3 vorgestellten Verfahren zur Realisierung des Datenflusses, bei dem prozessglobale Variablen verwendet werden (so wie auch bei ADEPT vorgesehen).

Alle bei der Simulation anfallenden Lastdaten (vgl. Anhang C.2) werden in einer Datei gespeichert. Diese kann nun mit Werkzeugen analysiert werden, die auf Verfahren basieren, welche in Abschnitt 9.3 noch vorgestellt werden. So können für eine Komponente oder mehrere Komponenten gemeinsam der Lastverlauf über die Zeit, die Last nach Beendigung der Einschwingphase und die zugehörigen Konfidenzintervalle berechnet werden.

9.2.3.2 Realisierung des Simulationsprogramms

Die Kernkomponente des Simulationsprogramms bildet eine Ereignisliste, in der alle schon bekannten, noch auszuführenden Aktionen zeitlich sortiert gespeichert werden. Solch eine Aktion kann z.B. sein, dass der Benutzer 24 zum Zeitpunkt 145,234 sec einen Eintrag aus seiner Arbeitsliste auswählt, dass zu einem bestimmten Zeitpunkt ein Aktivitätenprogramm von einem bestimmten Benutzer gestartet wird, oder dass es beendet wird.

In der Hauptschleife des Simulationsprogramms wird ständig der aktuellste (und damit vorderste) Eintrag aus der Ereignisliste entfernt und verarbeitet. Das heißt, die Ausführung der entsprechenden Aktion wird simuliert. Dabei können neue Einträge der Ereignisliste generiert werden: So entsteht z.B. bei der Abarbeitung des Eintrags „Aktivitätenprogramm starten“ ein neuer Eintrag „Aktivitätenprogramm beenden“, dem eine spätere Zeit zugeordnet ist. Außerdem werden alle für die Verarbeitung eines Eintrags benötigten Aktionen und Kommunikationen (zusammen mit dem zugehörigen Zeitpunkt und den betroffenen Komponenten) protokolliert. Auf Basis dieser Information kann später der Lastverlauf analysiert werden. Ist die Verarbeitung eines Eintrags abgeschlossen, so wird der nächste Eintrag verarbeitet. Da diesem ein späterer Zeitpunkt zugeordnet ist, springt die Simulation von Zeitpunkt zu Zeitpunkt. Es findet also keine kontinuierliche Bearbeitung über die Zeit statt, sondern es

werden nur die für die Simulation relevanten Zeitpunkte betrachtet. Deshalb läuft die Simulation viel schneller ab, als die reale Zeit. So können sehr viele sehr umfangreiche Simulationsläufe durchgeführt werden, was notwendig ist, um die geforderte Genauigkeit zu erreichen.

9.3 Auswertung der Simulationsergebnisse

Die Auswertung der Simulationsergebnisse soll an dem in Abschnitt 1.2.2 eingeführten Beispiel des Kreditantragsbearbeitungs-WF erläutert werden. Zur Erinnerung ist dieser WF in Abb. 9.8 nochmals dargestellt. Die bei der Simulation dieses WF berechneten Simulationsdaten sind im Anhang C.2 aufgelistet. Im Folgenden soll exemplarisch am Beispiel einiger Werte erläutert werden, wie die Simulationsergebnisse ausgewertet werden.

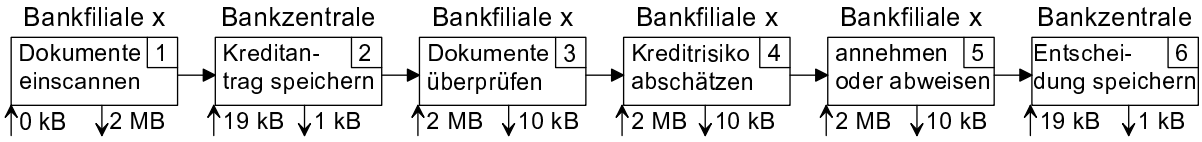


Abbildung 9.8 Beispiel-WF Kreditantragsbearbeitung aus Abb. 1.3.

9.3.1 Mittelwertbildung

Um genaue Ergebnisse zu erhalten, wurde die Simulation des Kreditantrags-WF (jeweils mit 39000 WF-Instanzen, vgl. Anhang C.1) 10000 Mal durchgeführt. Im Anhang C.2 sind die durch die Simulation ermittelten Lastmittelwerte angegeben. Die Ergebnisse der einzelnen Simulationsläufe werden ausschließlich zur Berechnung der Varianz und der Konfidenzintervalle benötigt (siehe Abschnitt 9.3.3) und sind aus Platzgründen im Anhang C.2 nicht aufgeführt. Für den WF-Server der Bankzentrale ergibt sich im nicht verteilten Fall eine durchschnittliche Last von 1958,77 kB/sec (vgl. Anhang C.2.1). Das heißt, die Netzwerkkarte des Servers muss diese Datenmenge umsetzen. Bei einem WF-Server ist außerdem die durch die auszuführenden Aktionen (z.B. Arbeitsliste aktualisieren, Aktivität starten) entstehende durchschnittliche Last interessant. So muss der Server 14,61 Aktionen pro Sekunde ausführen. Im Anhang C.2 ist die Last auch noch nach den einzelnen Aktionen aufgeschlüsselt angegeben.

Zusätzlich zur Mittelung über die Simulationsläufe findet zur Berechnung der Last (Aufwand pro Zeit) eine Mittelung über ein Zeitintervall statt. Dieses Zeitintervall ist die Phase, in der sich die Simulation in einem eingeschwungenen Zustand befindet. Die im Anhang C.2 und in diesem Kapitel angegebenen Lastwerte stehen stets für die durchschnittliche Last im eingeschwungenen Zustand. Wie der Zeitpunkt, ab dem der eingeschwungene Zustand erreicht ist, ermittelt werden kann, wird im nachfolgenden Abschnitt erläutert. Durch die Mittelwertbildung erhält man Ergebnisse, die sehr dicht am Erwartungswert der zu schätzenden Zufallsvariable liegen. Dies gilt insbesondere, da bei allen Simulationen 10000 Simulationsläufe – und damit sehr viele – gemittelt werden. Wie gut die Ergebnisse tatsächlich sind, wird in Abschnitt 9.3.3 untersucht.

9.3.2 Ermitteln der transienten Phase

Zu Beginn eines Simulationslaufs befinden sich noch keine WF-Instanzen im simulierten WfMS. Im Laufe der Simulation werden die WF-Instanzen „gestartet“. Ab einem gewissen Zeitpunkt verlas-

sen gleich viele Instanzen das System, wie neu gestartet werden, so dass ab diesem Zeitpunkt eine Sättigung der Last eintritt. Das System befindet sich nun in einem eingeschwungenen (stationären) Zustand, wohingegen die Zeit davor als transiente Phase bezeichnet wird [Lan92]. Da für unsere Untersuchungen die durchschnittliche Last im eingeschwungenen Zustand relevant ist, muss der Zeitpunkt ermittelt werden, ab dem eine Sättigung der Last eintritt. Dann können die „Messwerte“, die aus der transienten Phase stammen, bei der Mittelwertbildung ignoriert (verworfen) werden (vgl. [GMWW99]). Im vorliegenden Abschnitt wird untersucht, wie die Ausdehnung der transienten Phase ermittelt werden kann.

Um den Beginn der stationären Phase zu ermitteln, wird das Verfahren von Welch [Lan92, Wel83] verwendet. Bei diesem Verfahren wird die Ausdehnung der transienten Phase anhand einer graphischen Darstellung des Lastverlaufs ermittelt, so dass die entsprechenden Messwerte verworfen werden können. Die Kurve, die den Lastverlauf repräsentiert, wird zu diesem Zweck geglättet, so dass der Beginn der Sättigung leicht erkannt werden kann. Die dafür notwendige Vorgehensweise wird nun am Beispiel des Kreditantrags-WF für die Kommunikationslast des zentralen WF-Servers erläutert.

Betrachtet man für eine einzelne Beobachtung (Simulationslauf) alle Messwerte, so erhält man eine sehr „zackige“ Kurve. Anhand dieser ist es nicht möglich, zu erkennen, wann eine Sättigung der Last eintritt. Bei unserer Simulation können keine einzelnen Messwerte betrachtet werden, da die Simulation lediglich ermittelt, wann welche Datenvolumina übertragen werden oder welche Aktionen stattfinden. Es wird nicht die Last zu einem bestimmten Zeitpunkt gemessen. Um diese zu erhalten, müssen die Ereignisse eines sehr kleinen Zeitintervalls aufaddiert und durch die Intervallgröße dividiert werden. Der daraus für den WF-Server resultierende Lastverlauf ist in Abb. 9.9 dargestellt.

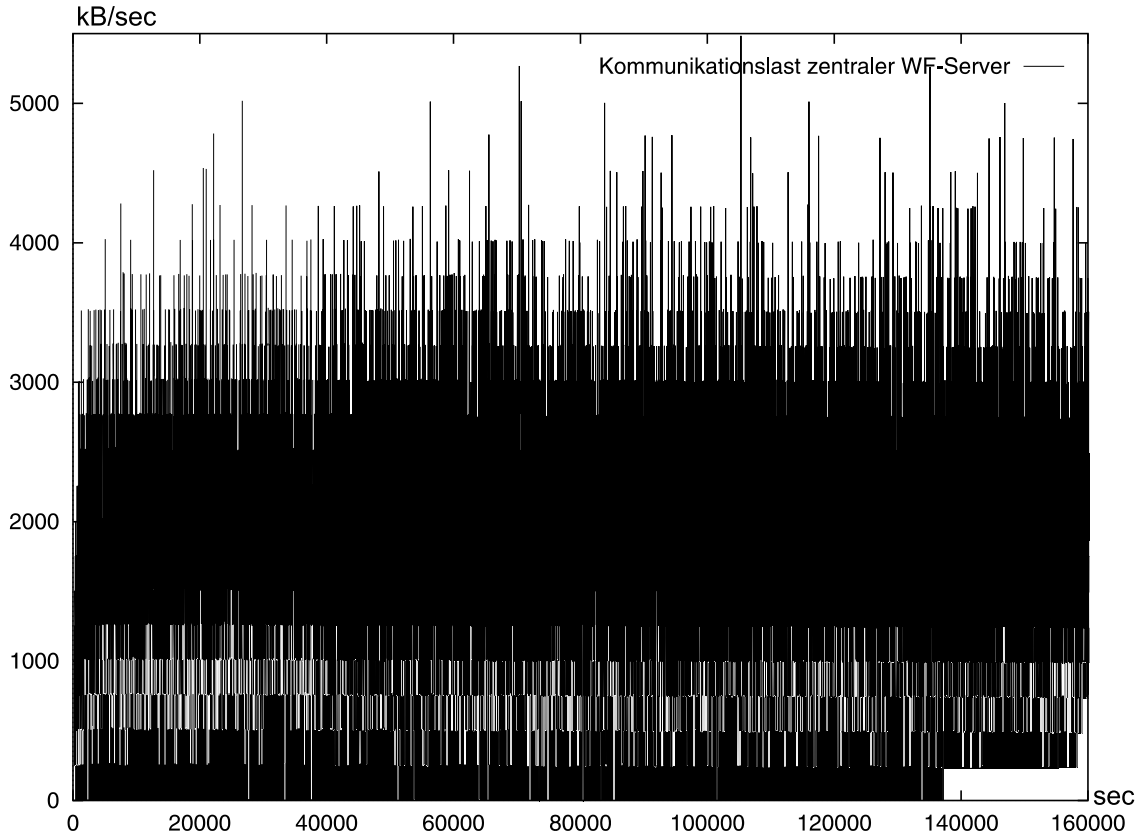


Abbildung 9.9 Lastverlauf für einen einzelnen Simulationslauf und 20000 Zeitintervalle.

Um die Kurve zu glätten, kann für jedes Intervall der Mittelwert der Last mehrerer Simulationsläufe gebildet werden. Das Ergebnis der Mittelung von 10000 Simulationsläufen ist in Abb. 9.10 dargestellt. Die Kurve ist schon wesentlich glatter. Im Allgemeinen lässt sich damit der Beginn der stationären Phase aber immer noch nicht zuverlässig erkennen. Da im vorliegenden Fall eine sehr große Anzahl von Simulationen durchgeführt wurde, lässt sich dieser Zeitpunkt jedoch auch schon in Abb. 9.10 recht gut schätzen.

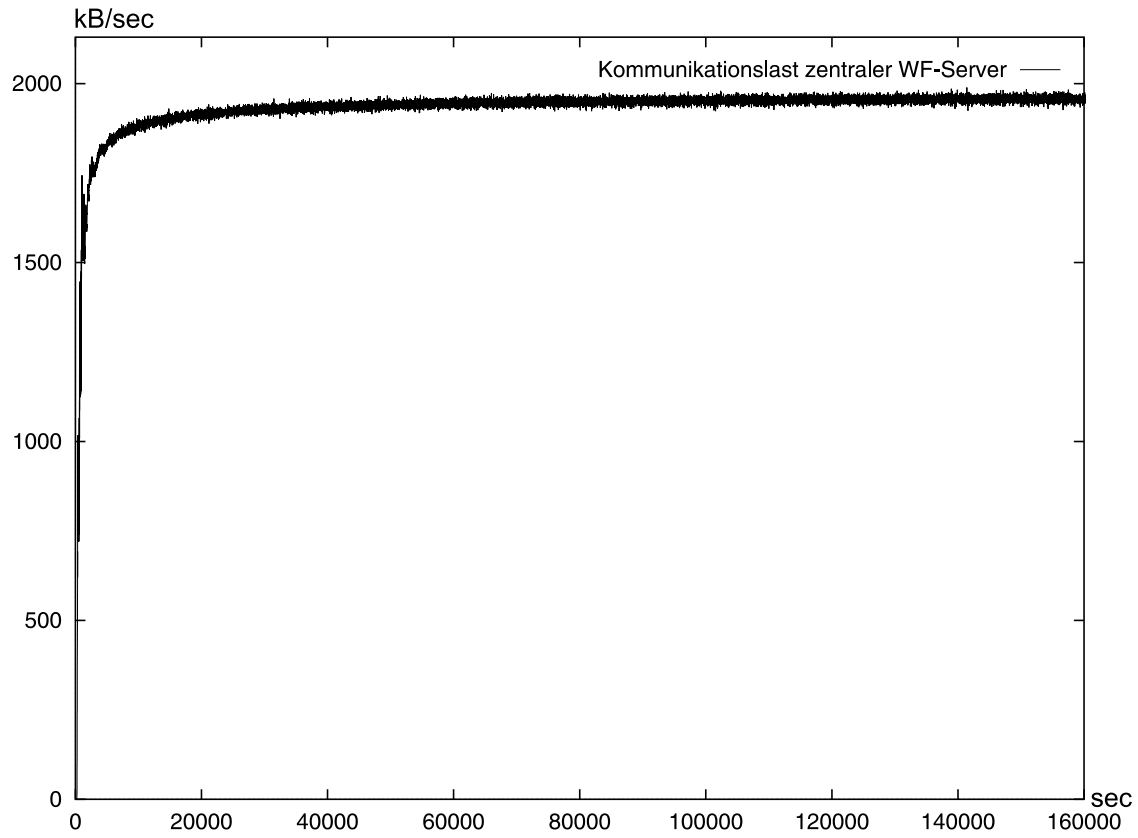


Abbildung 9.10 Lastverlauf für 10000 Simulationsläufe und 20000 Zeitintervalle.

Zur weiteren Glättung der Kurve schlägt Welch vor, den Mittelwert über mehrere Messwerte (Zeitintervalle) zu bilden. In der Originalarbeit [Wel83] wird für alle Intervalle i der Mittelwert der Last der k benachbarten Intervalle gebildet, so dass sich in unserem Beispiel wiederum 10000 Intervalle ergeben würden. Um im Simulationsprogramm weniger Intervalle verwalten zu müssen, fassen wir aber jeweils k Intervalle zu einem größeren Intervall zusammen. Dies führt ebenfalls zur Glättung der Kurve und es müssen in dem Beispiel nur 100 Zeitintervalle verwaltet werden. Dass insgesamt weniger Intervalle betrachtet werden, hat keinen Einfluss auf das optische Erkennen der Ausdehnung der transienten Phase, da 100 Intervalle immer noch ausreichen, um die Kurve nicht „eckig“ erscheinen zu lassen. Wie in Abb. 9.11 erkennbar ist, erreicht die Kurve ungefähr zum Zeitpunkt $t = 120000$ sec ihre Sättigung, weshalb alle davor liegenden Messwerte verworfen werden.

Die Vorteile des Verfahrens von Welch sind, dass es einfach und intuitiv ist. Anhand der graphischen Darstellung des Lastverlaufs erkennt man leicht, inwieweit die Abschätzung richtig und sicher ist. Deshalb ist das Verfahren sehr zuverlässig. In [Lan92] werden auch Verfahren vorgestellt, mit denen die Anzahl der zu löschenden Messwerte durch eine Berechnung ermittelt wird (Verfahren von Heidelberger und Welch, Verfahren von Kelton und Law) oder zur Analyse, ob eine transiente Phase vor-

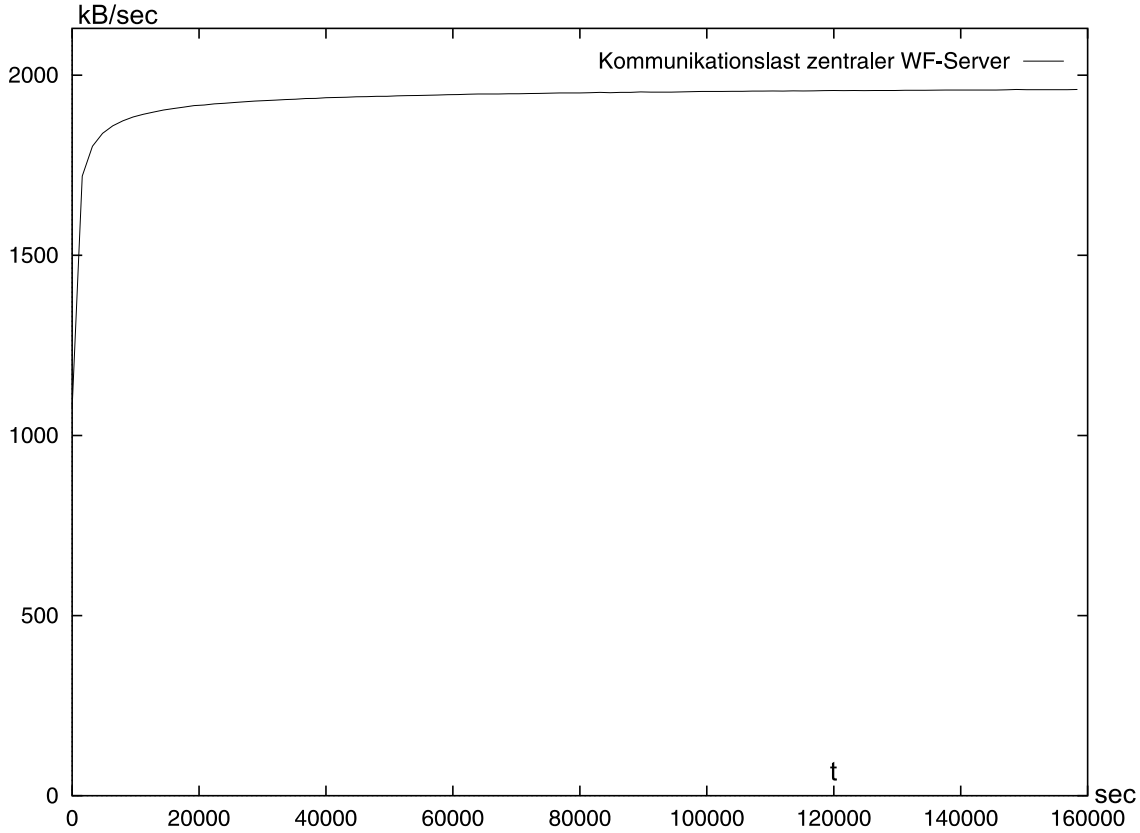


Abbildung 9.11 Lastverlauf für 10000 Simulationsläufe und 100 Zeitintervalle.

handen ist[1] (Test von Schruben, Test von Schruben, Singh und Thierney). Diese Verfahren erfordern aber allesamt die Vorgabe von zahlreichen Schätzwerten. Da diese bei falscher Schätzung zu falschen Analyseergebnissen führen können, sind die Verfahren im vorliegenden Fall wenig zuverlässig. Deshalb ist ihnen das Verfahren von Welch vorzuziehen.

9.3.3 Berechnung von Konfidenzintervallen

Es macht nur dann Sinn, Simulationen durchzuführen, wenn die Zuverlässigkeit der ermittelten Werte eingeschätzt werden kann. Um die Genauigkeit eines Mittelwertes anzugeben, kann prinzipiell die Varianz [Lan92] verwendet werden. Es ist aber wesentlich aussagekräftiger, wenn aus dieser ein Konfidenzintervall [BS89, Lan92, Wel83] berechnet wird. Zu diesem Zweck wurde bei allen Simulationen das 90%-Konfidenzintervall für den Erwartungswert verwendet. Dieses gibt das Intervall an, in dem sich der zu schätzende Wert (Erwartungswert der Zufallsvariablen) mit einer Wahrscheinlichkeit von 90% befindet. Wegen des großen Stichprobenumfangs (10000 Simulationsläufe) kann zu dessen Berechnung die Standardnormalverteilung verwendet werden (anstelle der Student'schen t-Verteilung) [Wel83].

[1] Wie schon zu Beginn dieses Abschnitts erläutert wurde, ist im vorliegenden Fall klar, dass eine transiente Phase existiert. Die Verfahren könnten aber verwendet werden, um, nach dem Löschen der ermittelten transienten Phase, zu berechnen, ob der Verlauf immer noch eine transiente Phase enthält. In diesem Fall müssten weitere Messwerte verworfen werden.

Im Anhang C.2 sind für alle Mittelwerte die 90%-Konfidenzintervalle aufgeführt. Dieses beträgt z.B. für die Kommunikationslast des Servers der Bankzentrale im nicht verteilten Fall (Anhang C.2.1) [1958,63 kB/sec, 1958,90 kB/sec] bei einem Mittelwert von 1958,77 kB/sec. Relativ ausgedrückt ergibt sich damit das Intervall: Mittelwert $\pm$ 0,0070%. Das Konfidenzintervall gibt an, wie genau der Mittelwert den Erwartungswert der Zufallsvariablen schätzt. Ein kleines Intervall steht dabei für eine hohe Genauigkeit. Ein solches kann erreicht werden, indem mehr Simulationsläufe durchgeführt werden. Die in diesem Kapitel vorgestellten Simulationen wurden jeweils so häufig wiederholt, dass die 90%-Konfidenzintervalle maximal um $\pm$0,1% um den Mittelwert variieren. Damit können die Simulationsergebnisse als sehr genau bezeichnet werden.

9.3.4 Relative Lastangaben

Die im Anhang C.2 angegebenen absoluten Lastwerte sind geeignet, um zu überprüfen, ob eine Komponente eines WfMS überlastet ist. Zu einem Vergleich der verschiedenen Verteilungsmodelle eignen sie sich aber nicht, da sie zu unübersichtlich sind. Deshalb werden in den nachfolgenden Abschnitten relative Lastangaben verwendet: Die Last der Server, Teilnetze und Gateways im zentralen Fall wird als Vergleichswert (100) definiert. Die Belastung bei den anderen Verteilungsmodellen wird relativ dazu angegeben. Ein Wert von 100 bedeutet also „dieselbe Last wie im zentralen Fall“ und nicht etwa „100% Auslastung der Komponente“.

Im Beispiel des Kreditantrags-WF beträgt die Gesamtlast aller Teilnetze im zentralen Fall 3907,6 kB/sec (vgl. Anhang C.2.1). Bei 31 Teilnetzen ergibt sich damit eine durchschnittliche Last pro Teilnetz von 126,1 kB/sec. Im verteilten Fall (Anhang C.2.2) beträgt die Last pro Teilnetz 63,5 kB/sec. Damit ergibt sich eine relative Last[2] von 63,5 kB/sec : 126,1 kB/sec $\cdot$ 100 = 50,4. Analog kann auch die relative Last der WF-Server, die Last pro WF-Server und die Gesamtlast der Gateways berechnet werden. Das dabei erzielte Ergebnis ist in Abb. 9.12a dargestellt. In Abb. 9.12b ist die relative Last dargestellt, die durch die Ausführung von Aktionen erzeugt wird. Auch diese wurde so normiert, dass sich für die Gesamtlast im zentralen Fall der Wert 100 ergibt. Die Last ist auch noch nach Typen von Aktionen aufgeschlüsselt (Werte neben den Balken in Abb. 9.12b). In dem vorliegenden Szenario ist die durch das Starten und Beenden von WF-Instanzen (3,3), durch das Starten und Beenden von Aktivitäteninstanzen (19,9) und durch Migrationen (0,0) erzeugte Last in beiden Verteilungsmodellen identisch. Die durch das Aktualisieren von Arbeitslisten erzeugte Last (gemessen in der Anzahl durchzuführender Aktionen pro Zeiteinheit) ist im verteilten Fall höher, da die Arbeitsliste eines Benutzers von mehreren Servern verwaltet wird. Dieser Aspekt wird in Abschnitt 9.4 ausführlich diskutiert. Die durch die Ausführung von Aktionen von den WF-Servern zu bewältigende Gesamtlast verteilt sich bei verteilter WF-Steuerung auf die im WfMS vorhandenen Server. Dies sind im vorliegenden Szenario 31 Server (siehe Anhang C), so dass sich die durchschnittlich von einem Server zu bewältigende Last als 147,3 / 31 = 4,8 ergibt. Auch die Anzahl der bei einem Verteilungsmodell verwendeten WF-Server ist über dem jeweiligen „Lastbalken“ angegeben.

9.4 Simulation eines Klinik-Workflows

Um die Verteilungsmodelle zu vergleichen, wird zuerst ein möglichst „realitätsnaher WF“ mit einem ausgewogenen Verhältnis von unabhängigen und abhängigen Bearbeiterzuordnungen simuliert.

[2] Da die 90%-Konfidenzintervalle kleiner als der Mittelwert $\pm$0,1% sind, ist die Angabe der 1. Nachkommastelle noch gerechtfertigt.

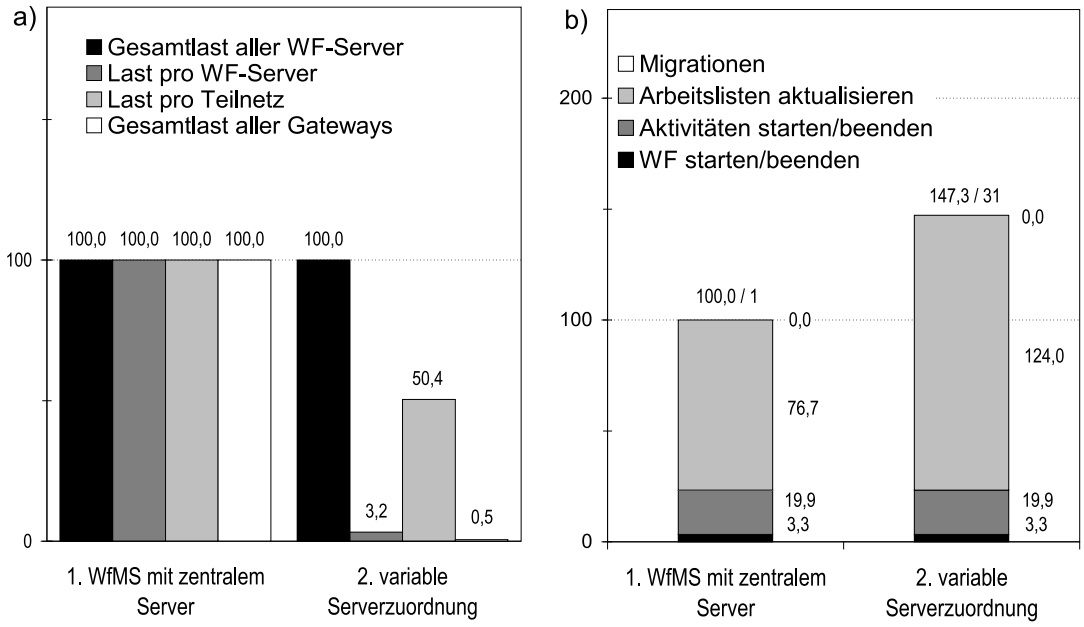


Abbildung 9.12 Relative Lastwerte für das Kreditantragsbeispiel: a) Kommunikation, b) Aktionen.

In Abschnitt 9.5 wird dann ein WF ohne abhängige Bearbeiterzuordnungen simuliert, um den Nutzen von statischen Serverzuordnungen untersuchen zu können. Variable Serverzuordnungen werden in Abschnitt 9.6 anhand eines WF mit überwiegend abhängigen Bearbeiterzuordnungen analysiert.

Im vorliegenden Abschnitt wird die Ausführung eines (stark vereinfachten) Klinik-WF untersucht. Wir betrachten dazu zunächst das gewählte Szenario und die Abbildung des WF auf die verschiedenen Verteilungsmodelle und anschließend die Simulationsergebnisse. Diese wurden größtenteils schon in [BD99b] publiziert.

9.4.1 Simulationsszenario

Der in Abb. 9.13 dargestellte Klinik-WF beginnt damit, dass ein Patient in die Ambulanz eingeliefert wird, wo er untersucht und eine Diagnose erstellt wird (Aktivität 1-6). Dazu muss eine Blutprobe im Labor untersucht werden (Aktivität 5). Falls notwendig, wird er auf einer von 3 möglichen Stationen aufgenommen, versorgt und behandelt (Aktivität 7-14). Nach der Entlassung des Patienten wird der WF in der Verwaltung fortgesetzt und schließlich beendet (Aktivität 15-21). Für die Bearbeitung wurden 35 Benutzer veranschlagt (2 Ambulanzärzte, 3 MTA, je Station 2 Ärzte und 6 Pflegekräfte, 4 Sachbearbeiter, 2 Laborassistenten). Jede Station, die Ambulanz, die Verwaltung und das Labor bilden jeweils einen eigenen Domain. Während der Simulationsdauer von 20 Tagen werden 500 Instanzen des Klinik-WF simuliert. Die konkreten Größen der Parameterdaten sind aus Platzgründen und zur Erhöhung der Übersichtlichkeit nicht angegeben. Da im Folgenden relative Lastangaben verwendet werden, haben sie auch keinen Einfluss auf das Simulationsergebnis. Eine Verdoppelung der Datenmenge führt bei allen Verteilungsmodellen zu einer Verdoppelung der Last und beeinflusst damit deren Vergleich nicht.

Die Abbildung des Klinik-WF auf die verschiedenen Verteilungsmodelle erfolgt wie in Tabelle 9.2 dargestellt:

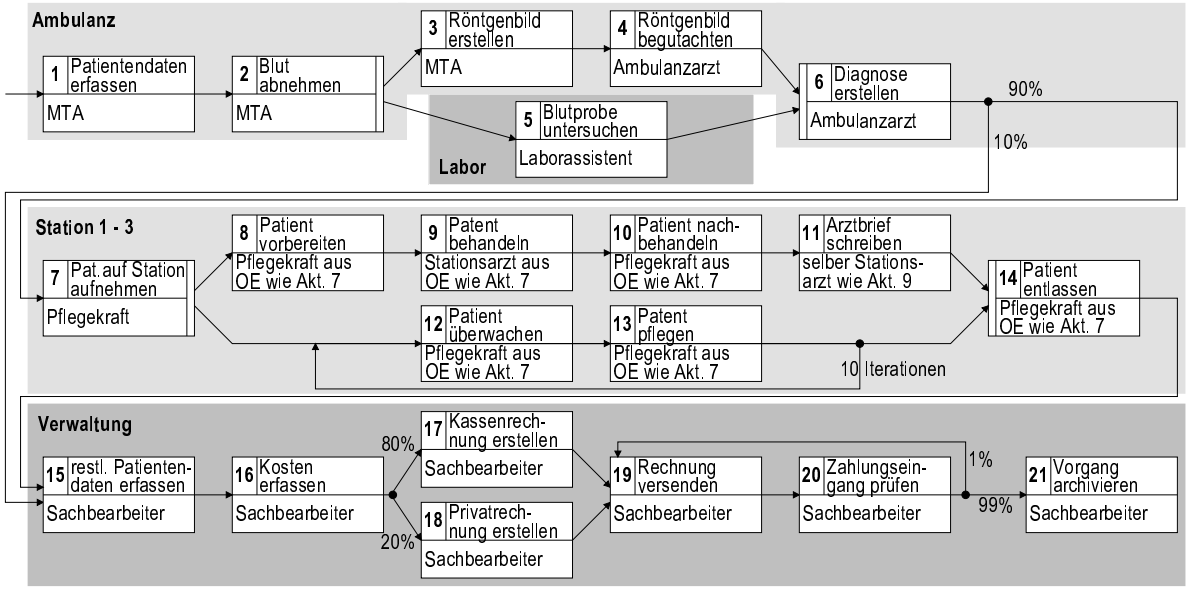


Abbildung 9.13 Struktur des simulierten Klinik-WF.

Modell 1: Zentraler Server Der WF-Server befindet sich im Teilnetz der Ambulanz. Mögliche Alternativen hierzu werden später diskutiert.

Modell 2: Server zufällig Der Server für eine gesamte WF-Instanz wird zufällig ausgewählt.

Modell 3: Keine Migration Das optimale Teilnetz für den WF-Server ist das der Verwaltung.

Modell 4: Keine globale Bearbeiterauflösung Ohne globale Bearbeiterauflösung müssen die Stationen 1-3 einen gemeinsamen Domain (ein Teilnetz mit nur einem WF-Server) bilden, damit die Aktivitäten 7-14 von den Benutzern aller 3 OE bearbeitet werden können. Deshalb verteilt sich die Gesamtlast bei diesem Ansatz auf insgesamt nur 4 Domains.

Modell 5: Migration und globale Bearbeiterauflösung Bei diesem Verteilungsmodell werden die Aktivitäten 7-11 und 14 vom Server der Verwaltung kontrolliert und nicht von dem einer Station, um Kommunikationskosten einzusparen.

Modell 6: Variable Serverzuordnung Variable Serverzuordnungen ermöglichen, dass die Aktivitäten 8-14 vom Server der Station kontrolliert werden, aus der die Bearbeiter stammen. Aktivität 7 wird vom Server der Ambulanz gesteuert, da vor ihrer Ausführung die betroffene Station noch nicht bekannt ist.

Modell 7: Bei der Anwendung Für jede Aktivität muss festgelegt werden, wo die zugehörige Anwendung angesiedelt ist. Dabei wurde – wann immer möglich – angenommen, dass die Anwendung im Teilnetz der zugehörigen Bearbeiter liegt. Andernfalls ergibt sich eine wesentlich höhere Belastung für die Teilnetze. Allerdings ist diese Annahme nicht für alle Aktivitäten möglich. Die Aktivitäten 1 und 15 (Patientendaten erfassen) verwenden dieselbe Anwendung, werden aber in unterschiedlichen OE bearbeitet. Aktivität 4 wurde (entsprechend ihrer Bearbeiter) dem Labor zugeordnet. Um unnötige Migrationen zu vermeiden, wurde angenommen, dass die Anwendungen der Aktivitäten 7-14 alle auf dem Server derselben Station installiert sind.

Modell 8: Voll verteilt Da es keine expliziten Server gibt, sind die in Tabelle 9.2 angegebenen 35 Server die Rechner der 35 Benutzer, die die WF-Steuerung übernehmen.

Verteilungsmodell	Anzahl Serv.	Anzahl Teiln.	Zuordnung der WF-Server zu den Aktivitäten
1. zentraler Server	1	6	Akt. 1-21: Ambulanz
2. Server zufällig	6	6	zufälliger Server für die gesamte WF-Instanz
3. keine Migration	6	6	Akt. 1-21: Verwaltung
4. keine globale Bearbeiterauflösung	4	4	Akt. 1-4, 6: Ambulanz, Akt 5: Labor, Akt. 7-14: Station, Akt. 15-21: Verwaltung
5. Migration und glob. Bearbeiterauflösung	6	6	Akt. 1-6: Ambulanz, Akt. 12-13: Station 1, Akt. 7-11, 14-21: Verwaltung
6. variable Server-zuordnung	6	6	Akt. 1-7: Ambulanz, Akt. 8-14: Station des Bearbeiters von Akt. 7, Akt. 15-21: Verwaltung
7. bei der Anwendung	6	6	Akt. 2-4, 6: Ambulanz, Akt. 5: Labor, Akt. 7-14: Station 1, Akt. 1, 15-21: Verwaltung
8. voll verteilt	35	6	Akt. 1-21: Rechner des Bearbeiters der Aktivität

Tabelle 9.2: Zuordnung der Aktivitäten zu den WF-Servern für die simulierten Verteilungsmodelle.

Bei der Auswahl des Prozesses für die Simulation wurde darauf geachtet, dass alle für die Verteilungsmodelle relevanten Aspekte berücksichtigt sind und eine ausgewogene Mischung erreicht wird. Würden bestimmte Aspekte fehlen (z.B. die Möglichkeit zur Migration), so würden verschiedene Verteilungsmodelle zusammenfallen. Wenn hingegen bestimmte Aspekte dominieren würden (z.B. ständiger Wechsel der OE), so wären die Ergebnisse nicht sehr aussagekräftig. Deshalb wurde ein WF gewählt, dessen Bearbeiter in verschiedenen OE angesiedelt sind (Ambulanz, Labor, Stationen, Verwaltung). Außerdem gibt es Aktivitäten (7-14), die je nach WF-Instanz in unterschiedlichen OE (Station 1-3) bearbeitet werden. Damit die Simulation eines WF allgemeine Aussagen über die Qualität der Verteilungsmodelle zulässt, sollte er Verzweigungen, Parallelität und Schleifen enthalten [GMWW99]. Auch dies ist bei dem simulierten Prozess gegeben. Schließlich wurde wie bei allen Simulationen darauf geachtet, dass die Belastung der Bearbeiter nur so groß ist, dass ihre Arbeitslisten nicht „überlaufen“.

9.4.2 Ergebnis der Simulation des Klinik-WF

In Abb. 9.14 ist die von den einzelnen Komponenten zu bewältigende Kommunikationslast dargestellt. Dabei wird für jedes Verteilungsmodell die Gesamtlast aller WF-Server, die Last pro WF-Server und die durchschnittliche Belastung der Teilnetze angegeben. Für die Gateways wird nicht die Belastung pro Gateway, sondern die insgesamt umgesetzte Last angegeben, weil Letztere der ggf. über ein WAN zu kommunizierenden Datenmenge (und damit den entstehenden Kosten) entspricht.

Bei den Verteilungsmodellen ohne Migration (2. und 3.) ist die Gesamtkommunikationslast der WF-Server identisch mit dem zentralen Fall, ansonsten ist sie höher. Dabei fällt auf, dass sie bei einem voll verteilten WfMS (8.) extrem hoch ist. Die Last pro Server ist bei den Ansätzen 2 - 8 viel kleiner als für den zentralen Server. Die durchschnittliche Teilnetzlast ist meist ähnlich zum zentralen Fall, nur durch variable Serverzuordnungen (6.) wird sie deutlich reduziert. Die Belastung der Gateways ist bei Ansätzen mit der Möglichkeit zur Migration (4.-8.) generell niedriger.

In Abb. 9.15 ist die Belastung der WF-Server dargestellt. Dabei ist nicht die Kommunikationslast angegeben, sondern die Anzahl der pro Zeiteinheit auszuführenden Aktionen (z.B. Arbeitsliste aktualisieren). Die Lastwerte sind wie in Abb. 9.12 nach Typ der Aktion aufgeschlüsselt. Alle angegebenen

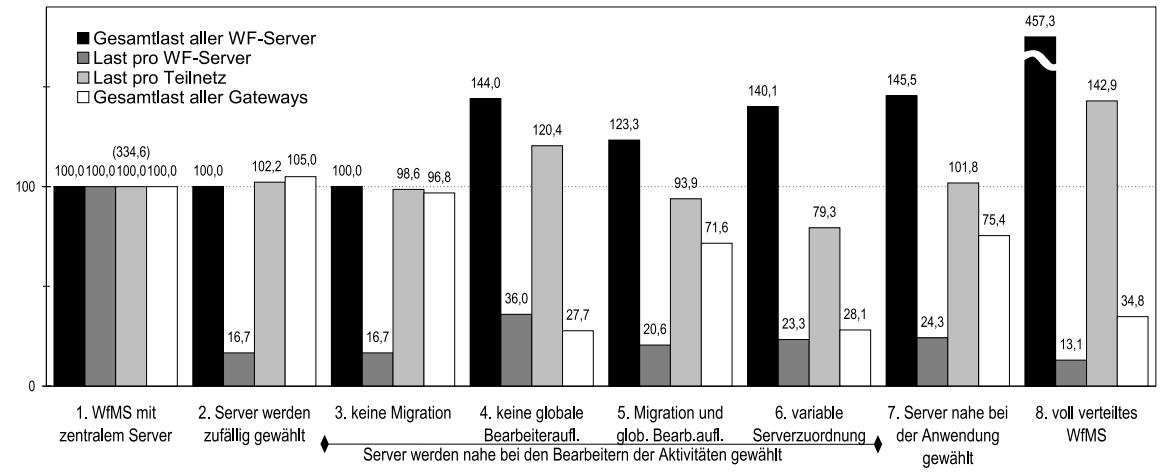


Abbildung 9.14 Kommunikationslast für die einzelnen Komponenten des WfMS.

Werte beschreiben die Gesamtlast der WF-Server. Bei Verteilungsmodellen mit mehreren Servern verteilt sich diese Last auf die WF-Server. Deshalb ist über den Balken wieder die Anzahl der WF-Server des simulierten Szenarios angegeben.

In Abb. 9.15 fällt auf, dass bei allen Verteilungsmodellen dieselbe Last für das Starten und Beenden von WF-Instanzen (0,5) bzw. von Aktivitäten (19,5) entsteht. Es war auch zu erwarten, dass dafür stets gleich viele Aktionen notwendig sind, da es sich jeweils um dieselbe Anzahl von WF-Instanzen bzw. Aktivitäten handelt. Die Last für das Aktualisieren von Arbeitslisten und für Migrationen ist bei einem voll verteilten WfMS (8.) auffallend groß. Generell sind bei Verteilungsmodellen mit Migrationen (4.-8.) die Werte in Abb. 9.15 kleiner, als die Gesamtkommunikationslast in Abb. 9.14, d.h. Migrationen erfordern zwar eine große Datenmenge, aber nicht die Ausführung von besonders vielen Aktionen. Deshalb sind sie für die WF-Server eher unkritisch, aber nicht für das Kommunikationssystem.

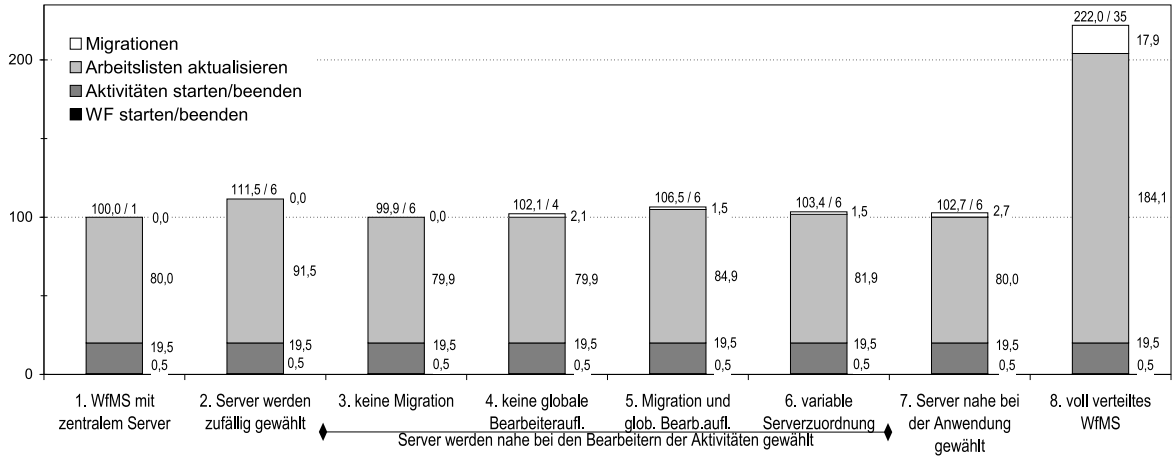


Abbildung 9.15 Durch von den WF-Servern auszuführende Aktionen erzeugte Last.

9.4.3 Diskussion der Simulationsergebnisse

Modell 1: Zentraler Server Die durchschnittliche Teilnetzlast hängt davon ab, in welchem Teilnetz der WF-Server angesiedelt ist. Konkret ergibt sich beim Teilnetz der Ambulanz die Last 100, bei den Stationen 101,1, bei der Verwaltung 98,6 und beim Labor 111,2. Es wurde also mit der Ambulanz ein „recht guter Fall“ ausgewählt (da es im zentralen Fall nur einen WF-Server im System gibt, kann dessen Lage nicht für einzelne WF-Typen optimiert werden). Die Last im Teilnetz des WF-Servers ist unabhängig von dieser Wahl stets 334,6. Diese hohe Last führt schnell zur Überlastung dieses Teilnetzes. Die Kommunikationslast des WF-Servers ist unabhängig davon, in welchem Teilnetz er angesiedelt ist. Da es nur einen Server im System gibt, ist die Gesamtlast der Server identisch mit der Last pro Server. Diese Gesamtlast kann von anderen Verteilungsmodellen nicht unterboten werden, da im zentralen Fall keine Kommunikation zur Synchronisation (z.B. Migrationen) notwendig ist. Die Gateway-Last wird bei diesem Verteilungsmodell alleine von den 5 Gateways umgesetzt, die das Teilnetz des WF-Servers mit denen von Bearbeitern verbinden. Um die Belastung der Teilnetze mit jener der Gateways vergleichen zu können, sei noch angemerkt, dass die von den Gateways umgesetzte Datenmenge 44,3% der in den Teilnetzen insgesamt kommunizierten Datenmenge entspricht. Betrachtet man die durch die Ausführung von Aktionen entstehende Last, so fällt auf, dass der Großteil (80%) durch die Aktualisierung von Arbeitslisten entsteht. Da bei der Ausführung von solchen Aktionen nur ein vergleichsweise geringes Datenvolumen umgesetzt wird, fallen sie in Abb. 9.14 kaum ins Gewicht. Für die Belastung der WF-Server stellt die Verwaltung der Arbeitslisten aber einen beträchtlichen Aufwand dar.

Modell 2: Server zufällig Hier ist ebenfalls keine Synchronisation zwischen den WF-Servern notwendig, da die WF-Instanzen unabhängig voneinander ausgeführt werden und ein WF komplett von einem Server kontrolliert wird. Deshalb erreicht die Gesamtkommunikationslast den optimalen Wert 100. Diese Last verteilt sich gleichmäßig auf die 6 WF-Server. Wegen der zufälligen Wahl des Servers findet keine Optimierung seines Ortes statt, weswegen sich für die Netz- und Gateway-Last sogar ein schlechterer Wert als im zentralen Fall ergibt. Die WF-Server müssen mehr Aktionen durchführen, als im zentralen Fall. Diese Aktionen sind zusätzliche Aktualisierungen von Arbeitslisten. Diese fallen an, weil 6 (anstelle von 1) WF-Server die Arbeitslisten der Benutzer verwalten. Da eine Arbeitsliste nicht sofort aktualisiert wird, wenn sie sich ändert, sondern Änderungen eine gewisse Zeit gesammelt und auf einmal übertragen werden (vgl. Abschnitt 2.5.3 und 9.2.3.1), bieten sich deshalb nicht mehr so viele Möglichkeiten zum Zusammenfassen von Änderungen. (Von verschiedenen WF-Servern durchzuführende Änderungen können natürlich nicht zusammengefasst werden.)

Modell 3: Keine Migration Wird der Server bei den Bearbeitern platziert und sind keine Migrationen möglich, so erreicht die Kommunikationslast der WF-Server den minimalen Wert 100. Da für verschiedene WF-Typen unterschiedliche Server verwendet werden können, verteilt sich diese Last auf die 6 Server. In der Simulation fällt die gesamte Last natürlich für den Server der Verwaltung an, da nur ein WF-Typ simuliert wird. Trotz der niedrigen Gesamtlast ist das Ergebnis für die Netzlast nicht besonders gut, da der Ort des WF-Servers nur für die gesamte WF-Instanz, nicht aber für einzelne Aktivitäten, optimiert werden kann. Deshalb befindet sich der Server für die Aktivitäten 1-14 im falschen Teilnetz. Betrachtet man die durch die Ausführung von Aktionen entstehende Last für die WF-Server, so ist diese mit dem zentralen Fall fast identisch. Die Last verteilt sich allerdings auf 6 WF-Server.

Modell 4: Keine globale Bearbeiterauflösung Weil zum Teilnetz der Bearbeiter jeder Aktivität migriert werden muss, entstehen nicht rentable Migrationen (z.B. wegen Aktivität 5 ins Labor). Deshalb resultiert eine recht hohe Kommunikationslast für die WF-Server. Da sich diese auf nur 4 Server ver-

teilt, ergibt sich bei diesem Ansatz die höchste Last pro Server. Aus denselben Gründen entsteht auch eine hohe Last pro Teilnetz. Prinzipiell wäre zwar vorstellbar, für den „zusammengesetzten Domain" der Stationen 3 Teilnetze zu verwenden, es entstünde aber keine Verbesserung für das Teilnetz des Servers. Dieses Teilnetz würde mit der gesamten Kommunikation des Domains belastet, da der WF-Server an jeder Kommunikation des Domains beteiligt ist. Der Grund dafür ist, dass die Clients der 3 Teilnetze ausschließlich mit diesem Server kommunizieren (wegen der lokalen Bearbeiterauflösung), und dass der Server die gesamte Kommunikation mit anderen Teilnetzen (Migrationen) abwickelt. Die Gesamtlast der Gateways ist bei diesem Ansatz besonders niedrig, da die 3 Stationen ein gemeinsames Teilnetz verwenden, so dass zwischen ihnen keine Kommunikation über ein Gateway stattfindet. Diese Last wird aber von nur 6 – und nicht wie bei den anderen Verteilungsmodellen 15 – Gateways umgesetzt. Bei den auszuführenden Aktionen kommt gegenüber dem zentralen Fall noch der Aufwand für die Migrationen hinzu. Dieser ist mit einem Anteil von ca. 2% an der Gesamtlast aber sehr gering.

Modell 5: Migration und globale Bearbeiterauflösung Die Kommunikationslast für die WF-Server ist kleiner als beim vorherigen Ansatz, da auf nicht rentable Migrationen verzichtet werden kann. Dasselbe gilt für die Anzahl der von den Servern auszuführenden Migrations-Aktionen (1,5 statt 2,1 bei 4.). Nicht rentable Migrationen sind z.B. die Migration zur Aktivität 5 und die zu einer Station wegen den Aktivitäten 7-11 und 14. Dass letztgenannte Aktivitäten der Verwaltung zugeordnet werden, spart für einen Großteil der WF-Instanzdaten eine zusätzliche Migration. Für die (mehrfach ausgeführten) Aktivitäten 12 und 13 ist es hingegen rentabel, sie vom Server einer Station kontrollieren zu lassen, da dann in $1/3$ der Fälle die Bearbeitung auf dieses Teilnetz beschränkt ist. Die Migrationskosten für den entsprechenden Teil der WF-Instanzdaten sind niedriger, als die durch die ständige Verwendung des falschen Servers entstehenden Kosten. Durch die Migrationen ist die Gesamtlast der Server natürlich größer als bei den Ansätzen 1, 2 und 3. Daraus ergibt sich eine höhere Last pro Server als bei verteilten Modellen ohne Migration. Sie ist aber deutlich niedriger als beim vorherigen Ansatz. Die Netzlast wird durch den vorliegenden Ansatz reduziert; die Last pro Teilnetz ist niedriger als bei den bisher beschriebenen Verteilungsmodellen. Bei der gewählten Verteilung sind etwas mehr Aktionen für das Aktualisieren der Arbeitslisten notwendig (84,9 gegenüber 80,0 im zentralen Fall), da wie bei 2. für die Arbeitsliste eines Benutzers mehrere WF-Server zuständig sein können. Dies gilt insbesondere für die Benutzer aus den Stationen 1-3, da für sie der Server der Station 1 (Aktivität 12-13) und der Server der Verwaltung (Aktivität 7-11, 14) zuständig ist (vgl. Tabelle 9.2).

Modell 6: Variable Serverzuordnungen Weil alle WF-Instanzdaten zusätzlich zum Server der entsprechenden Station migrieren müssen (Aktivität 8-14), ist die Gesamtkommunikationslast und die Kommunikationslast pro Server größer als bei statischer Serverzuordnung. Da keine nicht rentablen Migrationen ausgeführt werden, sind diese Werte aber immer noch besser als bei den Ansätzen 4 und 7. Durch die variable Serverzuordnung ergibt sich das beste Ergebnis für die Belastung der Teilnetze. Das Ergebnis bei statischer Serverzuordnung (5.) ist 18% schlechter, alle anderen Ansätze erzeugen mindestens 24% mehr Last. Dies zeigt, dass variable Serverzuordnungen und das Ermitteln der optimalen Verteilung adäquate Mittel sind, um die Netzlast zu reduzieren. Dies bezieht sich sowohl auf die durchschnittliche Netzlast, wie auch auf die Belastung der Gateways. Die durch die Aktionsausführung entstehende Last ist ähnlich (sogar etwas kleiner) wie bei statischen Serverzuordnungen. Sie stellt also kein Argument gegen die Verwendung von variablen Serverzuordnungen dar.

Modell 7: Bei der Anwendung Die unrentablen Migrationen (z.B. nach Beendigung der Aktivität 1) führen – verglichen mit den anderen Ansätzen, bei denen die Server geeignet gewählt werden – zu der höchsten Gesamtkommunikationslast. Der Wert für die Netzlast ist nicht ganz so schlecht, da für die meisten Aktivitäten der optimale Server verwendet wird. Allerdings wirken sich auch hier die

hohen Migrationskosten negativ aus. Der Aufwand für die Ausführung von Aktionen ist ähnlich wie bei den anderen Ansätzen mit Migrationen.

Modell 8: Voll verteilt Da es keine (expliziten) Server gibt, ist die in Abb. 9.14 für die Server angegebene Kommunikationslast die Gesamtlast aller Arbeitsstationen der Benutzer („Mini-Server"). Dieser Wert ist viel größer, als bei den bisher diskutierten Verteilungsmodellen, da die komplette WF-Instanz nach Beendigung jeder Aktivität migriert (außer sie wird vom selben Benutzer wie ihr Vorgänger bearbeitet). Diese Gesamtlast verteilt sich auf die Arbeitsstationen der 35 Benutzer. Die Last pro Rechner ist kleiner als bei den Ansätzen mit Servern. Da aber die Arbeitsplatzrechner der Benutzer und nicht leistungsstarke Server damit belastet werden, ist der Wert vergleichsweise hoch. Die Belastung der Gateways ist ziemlich niedrig, da ein WF stets in das Teilnetz der Station migriert wird (Aktivität 7-14), zu der die entsprechenden Bearbeiter gehören (vgl. variable Serverzuordnungen). Migrationen zwischen Bearbeitern derselben OE finden lokal in einem Teilnetz statt, was die Gateways nicht belastet, aber zu der höchsten Belastung der Teilnetze führt. Bei der durch die Aktionsausführung entstehenden Last fällt auf, dass der Aufwand für das Aktualisieren der Arbeitslisten extrem groß ist. Das liegt daran, dass jeder der 35 Mini-Server Arbeitslisten von quasi jedem anderen Benutzer aktualisieren muss. Deshalb können Änderungen fast nie zusammengefasst werden. Außerdem muss eine große Anzahl von Aktionen für Migrationen ausgeführt werden, da wie schon erwähnt sehr viele Migrationen stattfinden.

9.5 Workflow mit unabhängigen Bearbeiterzuordnungen

In diesem Abschnitt wird die Simulation eines WF mit ausschließlich unabhängigen Bearbeiterzuordnungen beschrieben. Diese Simulation dient dazu, den Sinn der Verwendung von statisch festgelegten Migrationen zu untersuchen. Es wird wieder zunächst das Simulationsszenario eingeführt und dann das Simulationsergebnis vorgestellt und diskutiert.

9.5.1 Simulationsszenario

Wir betrachten eine Umgebung mit 4 OE (Verkauf, Fertigung, Versand und Buchhaltung) mit jeweils 10 Mitarbeitern. Jede OE verfügt über ein eigenes Teilnetz und ggf. über einen eigenen WF-Server. Es wurde ein sehr einfacher WF simuliert, der aus einer Sequenz von 40 Aktivitäten besteht. Dabei bilden die Aktivitäten 1...10, 11...20, 21...30 und 31...40 Blöcke, von denen 9 Aktivitäten zur selben OE gehören (in der oben angegebenen Reihenfolge) und jeweils eine Aktivität zu einer anderen OE gehört. Diese einzelnen Aktivitäten sind die Aktivitäten 5, 15, 25 und 35. So wird z.B. innerhalb des Blocks der Aktivitäten des Verkaufs bei der Fertigung angefragt, ob die gewünschte Maschinenkonfiguration überhaupt realisiert werden kann.

Im Folgenden wird die Abbildung des WF auf die verschiedenen Verteilungsmodelle beschrieben. Die Verwendung von variablen Serverzuordnungen macht dabei keinen Sinn, da in dem WF keine abhängigen Bearbeiterzuordnungen enthalten sind. Wird der WF-Server bei der Anwendung platziert, so wird wieder wohl wollend angenommen, dass sich die Anwendung jeweils im Domain der Bearbeiter der zugehörigen Aktivität befindet. Damit ergibt sich dieselbe Verteilung wie bei lokaler Bearbeiterauflösung (Modell 4), weshalb dieser Fall hier nicht gesondert betrachtet wird.

Modell 1: Zentraler Server Da die Benutzer aller OE dieselbe Anzahl von Aktivitäten bearbeiten, spielt es keine Rolle, in welchem Domain der WF-Server platziert wird. Für die Simulation wird der Server der Fertigung verwendet.

Modell 2: Server zufällig Für die gesamte WF-Instanz wird zufällig ein Server ausgewählt.

Modell 3: Keine Migration Auch hier wird der Server der Fertigung gewählt.

Modell 4: Keine globale Bearbeiterauflösung Da die Bearbeiterauflösung nur lokal erfolgt, muss zur Ausführung der Aktivitäten 5, 15, 25 und 35, die von einer anderen OE bearbeitet werden als der umgebende Block, migriert werden. Ansonsten werden die Blöcke vom Server der entsprechenden OE kontrolliert.

Modell 5: Migration und globale Bearbeiterauflösung Die Blöcke werden jeweils komplett vom Server der entsprechenden OE kontrolliert, da sich eine Migration wegen der einzelnen Aktivitäten 5, 15, 25 und 35 nicht lohnt.

Modell 6: Voll verteilt Die Rechner der 40 Bearbeiter des WF übernehmen die WF-Steuerung.

9.5.2 Simulationsergebnis

In Abb. 9.16 ist das Simulationsergebnis für die Kommunikationslast der einzelnen Komponenten dargestellt. Dabei fällt vor allem auf, dass bei Ansätzen mit Migrationen (4.-6.) die von den WF-Servern insgesamt umgesetzte Datenmenge recht groß ist (dies gilt insbesondere bei 4. und 6., den Verteilungsmodellen bei denen Migrationen erzwungen werden). Dahingegen ist die Belastung der Teilnetze und Gateways bei diesen Verteilungsmodellen kleiner als im zentralen Fall.

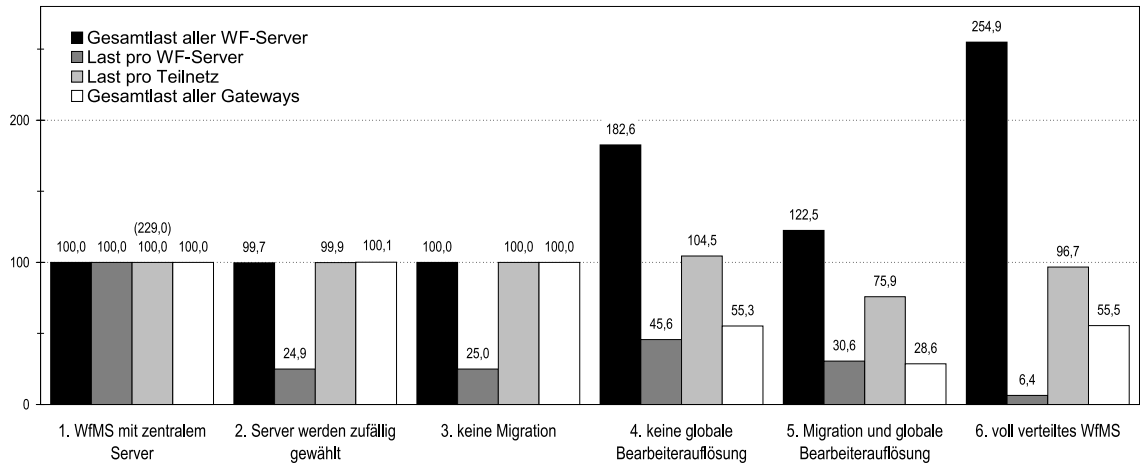


Abbildung 9.16 Kommunikationslast für die einzelnen Komponenten des WfMS.

Abb. 9.17 zeigt die durch die Ausführung von Aktionen erzeugte Last. Für das Starten und Beenden von WF-Instanzen und Aktivitäten fällt, wie in Abschnitt 9.4 erläutert, bei allen Verteilungsmodellen dieselbe Last an. Die Gesamtlast weicht nur bei 2. und 6. stark von der des zentralen Falls ab. Bei diesen Verteilungsmodellen entsteht hauptsächlich durch das Aktualisieren von Arbeitslisten eine höhere Anzahl von auszuführenden Aktionen.

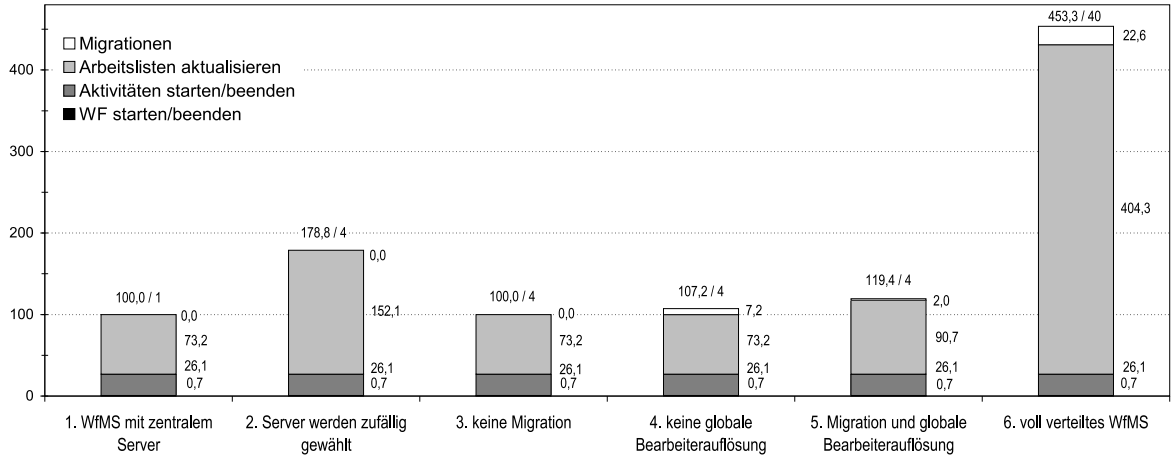


Abbildung 9.17 Durch von den WF-Servern auszuführende Aktionen erzeugte Last.

9.5.3 Diskussion der Simulationsergebnisse

Modell 1: Zentraler Server Die Last wurde wieder so normiert, dass sich für alle Komponenten der Wert 100 ergibt. Die Teilnetze werden wie immer bei Verwendung eines zentralen WF-Servers sehr ungleich belastet. Im vorliegenden Szenario beträgt die Last im Teilnetz des WF-Servers 229,0, während die Last in den anderen Teilnetzen durchschnittlich 57,0 ist. Die insgesamt von den Gateways umgesetzte Datenmenge beträgt bei dieser Simulation 42,8% der insgesamt in den Teilnetzen kommunizierten Datenmenge, so dass sowohl die Gateways wie auch die Teilnetze überlastet sein können. Bei der Ausführung von Aktionen entsteht der Hauptaufwand (73,2) wieder durch das Aktualisieren von Arbeitslisten.

Modell 2: Server zufällig Die Werte für die Kommunikationslast sind fast identisch mit denen des zentralen Falls, wobei sich die von den WF-Servern zu bewältigende Last allerdings auf 4 Server verteilt. Für das Aktualisieren der Arbeitslisten sind vergleichsweise viele Aktionen (152,1 anstatt 73,2) notwendig, da durch die verteilte Verwaltung der Arbeitslisten das Zusammenfassen von Änderungen erschwert wird.

Modell 3: Keine Migration Bei diesem Verteilungsmodell ergibt sich dasselbe Ergebnis wie im zentralen Fall, da alle WF-Server gleich gut geeignet sind und deshalb kein Optimierungspotential existiert. Die von den Servern zu bewältigende Last verteilt sich allerdings (rechnerisch) auf die 4 WF-Server.

Modell 4: Keine globale Bearbeiterauflösung Findet die Bearbeiterauflösung nur lokal statt, so muss stets in den Domain der Bearbeiter der aktuellen Aktivität migriert werden. Da die Migrationen zu den Aktivitäten 5, 15, 25 und 35 nicht rentabel sind, ergibt sich eine sehr hohe Kommunikationslast für die WF-Server und eine relativ große Anzahl von Migrations-Aktionen. Die Belastung der Teilnetze bzw. Gateways ist, trotz der nicht rentablen Migrationen, nur geringfügig größer bzw. sogar kleiner als im zentralen Fall.

Modell 5: Migration und globale Bearbeiterauflösung Da bei diesem Verteilungsmodell auf nicht rentable Migrationen verzichtet werden kann, ist die Belastung der WF-Server bzgl. Datenvolumen und Anzahl Migrations-Aktionen kleiner als bei den anderen Verteilungsmodellen mit Migrationen (4. und 6.). Für die Belastung der Teilnetze und Gateways ergeben sich eine deutliche Reduktion der Last

gegenüber dem zentralen Fall und die besten Werte aller Verteilungsmodelle. Dies zeigt, dass sowohl globale Bearbeiterauflösung wie auch Migrationen unbedingt notwendig sind, um die Belastung des Kommunikationssystems zu reduzieren, d.h. um ein effizientes WF-Management zu realisieren.

Modell 6: Voll verteilt Die Belastung der Mini-Server ist bei diesem Verteilungsmodell sehr groß, sowohl was die Kommunikationslast, als auch was die Anzahl der auszuführenden Aktionen angeht. Dies resultiert wie schon im Beispiel aus Abschnitt 9.4 daraus, dass jeder Rechner einen Teil der Arbeitsliste jedes Benutzers des WfMS verwaltet, und daraus, dass quasi nach Beendigung jeder Aktivität migriert werden muss. Die Belastung der Teilnetze und Gateways ist recht niedrig. Dies liegt daran, dass beim Starten einer Aktivität keine Kommunikationskosten mehr anfallen, da die benötigten Daten schon bei der Migration auf den Rechner des entsprechenden Benutzers transportiert wurden. Deshalb ist die Lage des WF-Servers stets optimal.

9.6 Workflow mit abhängigen Bearbeiterzuordnungen

Für die in diesem Abschnitt beschriebene Simulation wurde ein Prozess gewählt, wie er für viele Umgebungen typisch ist. Während in Abschnitt 9.4 kaum und in Abschnitt 9.5 keine Abhängigkeiten zwischen den Bearbeitern der einzelnen Aktivitäten unterstellt werden, wird nun der in der Praxis häufig auftretende Fall betrachtet, dass derartige Abhängigkeiten bestehen. Teile der Simulationsergebnisse wurden in [BD99a, BD00a] publiziert. Ein ähnliches Szenario findet sich in [BD99b].

9.6.1 Simulationsszenario

Abhängige Bearbeiterzuordnungen treten in zahlreichen Anwendungsbereichen auf. Enthält ein WF viele abhängige Bearbeiterzuordnungen, so sind für seine verteilte Ausführung variable Serverzuordnungen sehr gut geeignet. Wie groß der damit erzielbare Gewinn ist, soll anhand der Simulation eines WF aus dem medizinischen Bereich untersucht werden. Wir betrachten ein Szenario mit einer Ambulanz (4 Ärzte), in die ein Patient eingeliefert wird. Anschließend wird er auf eine von 5 Krankenstationen verlegt (je 2 Ärzte). Für die auf den Stationen auszuführenden Aktivitäten werden abhängige Bearbeiterzuordnungen verwendet, da für den Patienten nur Personal der betroffenen Station x zuständig ist. Allerdings wird während des Aufenthalts des Patienten auf der Station x noch eine Aktivität im Labor (1 MTA) ausgeführt. Damit ergibt sich für den WF die folgende Sequenz von Aktivitäten:
Aktivität 1-3 in der Ambulanz
Aktivität 4 durch einen Stationsarzt (nimmt Patient auf Station 1-5 auf)
Aktivität 5-9 durch einen Arzt dieser Station x
Aktivität 10 im Labor
Aktivität 11-15 durch einen Arzt der Station x

Der beschriebene WF wird wie folgt auf die Verteilungsmodelle abgebildet:

Modell 1: Zentraler Server Alle Aktivitäten werden vom Server der Station 1 kontrolliert. Die Server der anderen Stationen würden zum selben Simulationsergebnis führen; die Server der Ambulanz und des Labors führen zu höheren Kommunikationskosten.

Modell 2: Server zufällig Der Server für eine gesamte WF-Instanz wird zufällig ausgewählt.

Modell 3: Keine Migration Wie im zentralen Fall wird der gesamte WF vom Server der Station 1 kontrolliert.

Modell 4: Keine globale Bearbeiterauflösung Die Aktivitäten 1-3 werden vom Server der Ambulanz kontrolliert. Die im Labor bearbeitete Aktivität 10 wird von dessen WF-Server gesteuert. Die einer Station zugeordneten Aktivitäten werden vom Server des Domains „Stationen“ kontrolliert. Wie bei der Simulation aus Abschnitt 9.4 müssen die 5 Stationen einen gemeinsamen Domain bilden, da die Bearbeiterauflösung nur lokal erfolgt.

Modell 5: Migration und globale Bearbeiterauflösung Da sich eine Migration ins Labor wegen einer einzelnen Aktivität nicht lohnt, werden die Aktivitäten 1-3 vom Server der Ambulanz und die restlichen Aktivitäten vom Server der Station 1 kontrolliert.

Modell 6: Variable Serverzuordnungen Die Aktivitäten 1-4 werden vom Server der Ambulanz kontrolliert, die Aktivitäten 5-15 vom Server der jeweils involvierten Station x. Es ist nicht möglich, schon die Aktivität 4 diesem Server zuzuordnen, da die betroffene Station vor Ausführung von Aktivität 4 noch nicht feststeht.

Modell 7: Bei der Anwendung Bei diesem Verteilungsmodell wird wieder wohl wollend angenommen, dass alle Anwendungen im Domain der Bearbeiter der jeweiligen Aktivität installiert sind. Außerdem seien alle Aktivitätenprogramme mit Bearbeitern einer Station im Domain der Station 1 installiert, so dass zwischen diesen Aktivitäten keine Migrationen stattfinden. Damit ergibt sich die folgende Verteilung: Die Aktivitäten 1-3 werden vom Server der Ambulanz, die Aktivitäten 5-9 und 11-15 werden vom Server der Station 1 und die Aktivität 10 wird vom Server des Labors kontrolliert.

Modell 8: Voll verteilt Die WF-Steuerung wird von den Rechnern der 15 Bearbeiter des WF übernommen.

9.6.2 Simulationsergebnis

In Abb. 9.18 ist die relative Kommunikationslast der Komponenten des WfMS bei den verschiedenen Verteilungsmodellen dargestellt. Dabei fällt auf, dass bei den Verteilungsmodellen, bei denen Migrationen erzwungen werden (4., 7. und 8), sehr hohe Lastwerte auftreten. Werden statische Serverzuordnungen verwendet (5.), so ist das Ergebnis fast mit denen der Verteilungsmodelle ohne Migrationen (1.-3.) identisch. Durch variable Serverzuordnungen (6.) lässt sich die Belastung der Teilnetze und vor allem der Gateways drastisch reduzieren.

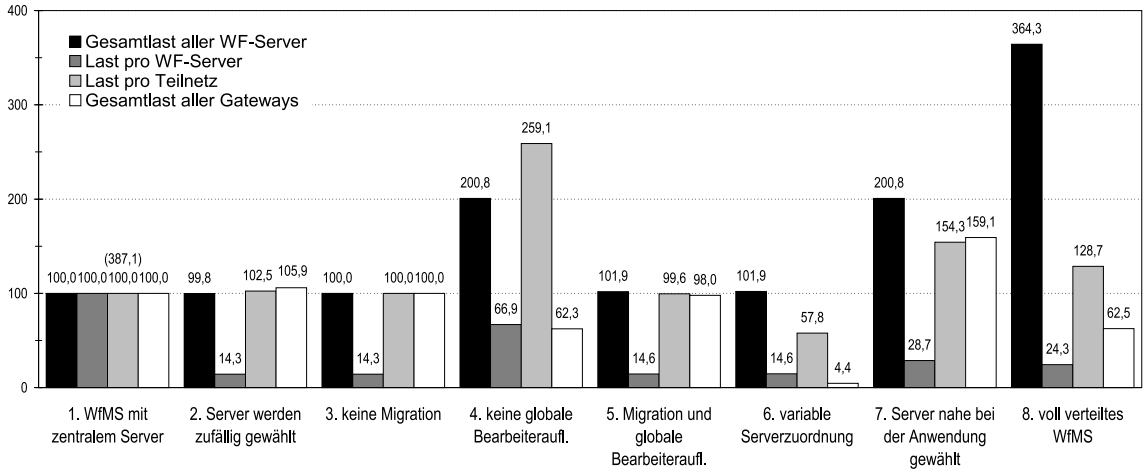


Abbildung 9.18 Kommunikationslast für die einzelnen Komponenten des WfMS.

Abb. 9.19 zeigt die bei den WF-Servern durch die Aktionsausführung erzeugte Last. Diese ist bei den Verteilungsmodellen 1. und 3.-7. sehr ähnlich. Lediglich, wenn die Server zufällig ausgewählt werden (2.) ist sie etwas größer, und bei dem voll verteilten Ansatz (8.) ist die Last extrem groß. Bei allen Verteilungsmodellen ergibt sich durch das Starten und Beenden von WF-Instanzen dieselbe Last 2,4 und durch das Starten und Beenden von Aktivitäteninstanzen die Last 35,6.

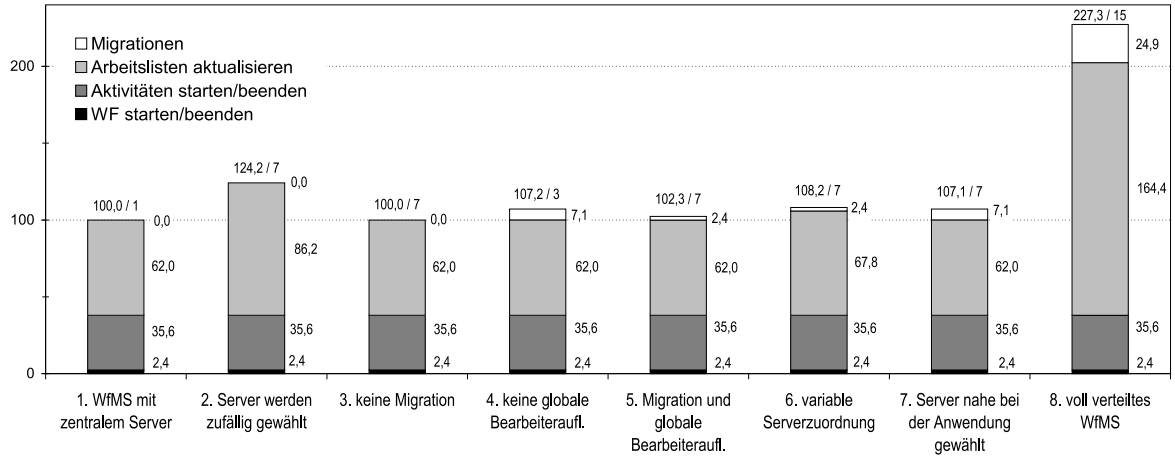


Abbildung 9.19 Durch von den WF-Servern auszuführende Aktionen erzeugte Last.

9.6.3 Diskussion der Simulationsergebnisse

Modell 1: Zentraler Server Die relative Last wurde wieder so normiert, dass sich für alle Komponenten der Wert 100 ergibt. Die Last in dem Teilnetz, in dem der WF-Server liegt, hat den Wert 387,1, während die anderen Teilnetze durchschnittlich nur mit 52,1 belastet werden. Das Verhältnis Gesamtkommunikationslast der Teilnetze zu Gesamtlast der Gateways beträgt in diesem Szenario 2,2.

Modell 2: Server zufällig Die mit dem zentralen Fall fast identische Gesamtkommunikationslast der WF-Server verteilt sich bei diesem Verteilungsmodell auf 7 Server. Die Belastung der Teilnetze und Gateways ist etwas erhöht, da außer den WF-Servern der Stationen auch die ungünstigeren Server der Ambulanz und des Labors verwendet werden. Da die Arbeitslisten eines Benutzers wieder durch mehrere Server verwaltet werden, müssen diese dabei mehr Aktionen ausführen, als im zentralen Fall.

Modell 3: Keine Migration Da dieselbe Verteilung verwendet wird, ergibt sich auch dieselbe Last wie im zentralen Fall. Allerdings verteilt sich diese rechnerisch auf 7 WF-Server.

Modell 4: Keine globale Bearbeiterauflösung Wegen der zahlreichen (erzwungenen) Migrationen entsteht eine hohe Kommunikationslast. Die Migrationen schlagen sich auch in der durch die Aktionsausführung erzeugten Last (7,1 für Migrationen) nieder. Da sich die hohe Kommunikationslast auf nur 3 Teilnetze verteilt, resultiert mit 259,1 eine sehr hohe Last pro Teilnetz. Die von den Gateways umgesetzte Datenmenge ist mit 62,3 vergleichsweise gering, da sich der WF-Server bei der Aktivitätenausführung stets im optimalen Teilnetz befindet. Außerdem bilden die Stationen 1-5 ein gemeinsames Teilnetz, so dass auch in die Ausführung der Aktivitäten 5-9 und 11-15 kein Gateway involviert ist.

Modell 5: Migration und globale Bearbeiterauflösung Die Belastung der WF-Server (sowohl Kommunikationen wie auch Aktionen) wird durch die einzelne Migration kaum erhöht. Allerdings

sind statische Serverzuordnungen im vorliegenden Szenario wenig geeignet, so dass die Belastung des Kommunikationssystems gegenüber dem zentralen Fall nur minimal reduziert werden kann.

Modell 6: Variable Serverzuordnungen Auch bei diesem Verteilungsmodell werden die WF-Server durch die einzelne Migration nur minimal stärker belastet, als im zentralen Fall. Die Belastung der Teilnetze und der Gateways kann aber drastisch reduziert werden. Die Belastung der Teilnetze wird annähernd halbiert (fast jede Kommunikation belastet nur noch ein Teilnetz, anstatt wie im zentralen Fall 2 Teilnetze). Kommunikation über ein Gateway findet äußerst selten statt (4.4), weil durch die variable Serverzuordnung für jede Aktivität der Server der betroffenen Station verwendet wird. In ADEPT wurde das Ziel verfolgt, die Belastung des Kommunikationssystems zu reduzieren. Dies ist, wie mit dieser Simulation eindrucksvoll gezeigt wurde, durch die Verwendung von variablen Serverzuordnungen gelungen. Es gibt also Szenarien, in denen variable Serverzuordnungen unverzichtbar sind, wenn ein bzgl. der Kommunikationskosten effizientes WF-Management realisiert werden soll.

Modell 7: Bei der Anwendung Die Verteilung ist identisch mit dem Modell 3, mit dem Unterschied, dass die Stationen weiterhin eigene Domains bilden. Deshalb ergibt sich dieselbe Gesamtlast für die WF-Server. Diese verteilt sich hier aber auf 7 (anstatt 3) WF-Server. Auch die Kommunikationslast verteilt sich auf die 7 Teilnetze und ist damit kleiner als beim Modell 3. Allerdings sind bei Aktivitäten, die von Bearbeitern einer Station ausgeführt werden, häufig 2 Teilnetze involviert (Station 1 des WF-Servers und Station x des Bearbeiters), so dass sich eine größere von den Gateways zu bewältigende Gesamtkommunikationslast als bei 3. ergibt.

Modell 8: Voll verteilt Wegen der zahlreichen Migrationen, und weil die Arbeitsliste eines Benutzers von vielen verschiedenen Mini-Servern aktualisiert werden muss, ergibt sich eine große Kommunikations- und Aktionslast für die WF-Server. Diese schlägt sich auch in einer hohen Last für die Teilnetze nieder. Lediglich die Belastung der Gateways ist moderat, da sich der WF-Server (ähnlich wie bei variablen Serverzuordnungen) stets im optimalen Domain befindet. Die Migrationen zwischen den Bearbeitern desselben Domains (z.B. zwischen den Aktivitäten 5-9) belasten die Gateways nicht.

9.7 Zusammenfassung und Diskussion

In diesem Kapitel wurden aus der Literatur bekannte Verteilungsmodelle klassifiziert und mit dem von ADEPT verglichen. Die entstehenden Klassen bilden die Basis für die Durchführung von Simulationen. Deren Ergebnisse wurden vorgestellt und diskutiert. Diese Ergebnisse sind sehr aussagekräftig, da die Konfidenzintervalle für die Erwartungswerte extrem klein sind, und weil die Unterschiede zwischen der bei den verschiedenen Verteilungsmodellen entstehenden Last groß ist. Mehrere Konzepte, die ausschließlich das Verteilungsmodell von ADEPT bietet, ermöglichen eine drastische Reduzierung der von den Teilnetzen und Gateways umzusetzenden Kommunikationslast. Dies sind namentlich die Verwendung von Migrationen in Kombination mit globaler Bearbeiterauflösung und die Verwendung von variablen Serverzuordnungen. Andere Verteilungsmodelle führen zwar in einzelnen Szenarien zu akzeptablen Ergebnissen, dafür aber in anderen zu einer hohen Last. Dagegen minimiert ADEPT stets die Last bzgl. der verwendeten Zielfunktion (hier: Minimierung der durchschnittlichen Belastung der Teilnetze), weil die Serverzuordnungen eben so gewählt werden, dass dies erreicht wird. Als weiteres Ergebnis der Simulationen bleibt festzuhalten, dass die durch Migrationen verursachte Mehrbelastung für die WF-Server klein ist. Dies gilt auch, wenn variable Serverzuordnungen verwendet werden. Insbesondere, wenn das für WF-Server relevante Maß der Anzahl der pro Zeiteinheit

durchzuführenden Aktionen zugrunde gelegt wird, ist die durch Migrationen verursachte Mehrbelastung fast vernachlässigbar. Aus diesen Gründen ist es essentiell, dass ein Verteilungsmodell (so wie ADEPT) eine Optimierung der Verteilung dadurch ermöglicht, dass eine WF-Instanz (abschnittsweise) von verschiedenen Servern kontrolliert werden kann.

Kapitel 10

Diskussion

Nachdem das vorherige Kapitel einen Überblick über verteilte WfMS bot, wird ADEPT$_{distribution}$ nun in diesem Zusammenhang diskutiert. Das Ziel des vorliegenden Kapitels ist es, aufzuzeigen, bei welchen Themen dieser Arbeit es sich um originäre Beiträge zum Stand der Forschung handelt. Dazu orientiert sich dieses Kapitel am Aufbau der Arbeit, so dass sich die Themen des Teils II im Folgenden jeweils als eigener Abschnitt wieder finden.

10.1 Verteilte Workflow-Steuerung

Im Abschnitt 9.1 wurden die aus der Literatur bekannten Systeme und Forschungsansätze für verteiltes WF-Management bereits ausführlich betrachtet und ADEPT$_{distribution}$ wurde mit den einzelnen Ansätzen verglichen. Dabei fällt auf, dass außer ADEPT$_{distribution}$ kein Ansatz das Ziel verfolgt, die Belastung des Kommunikationssystems durch die Wahl einer geeigneten Aufgabenverteilung zu minimieren. Unser Ansatz ist außerdem der einzige, der sowohl eine verteilte WF-Steuerung (mit Migrationen), als auch eine globale Bearbeiterauflösung vorsieht. Dies ist aber notwendig, um überhaupt Freiheitsgrade bei der Wahl des WF-Servers einer Aktivität zu haben. Bei allen anderen Verteilungsmodellen sind überhaupt keine Migrationen möglich (bei zentralem WF-Server, bei zufälliger Wahl des Clusters und bei Festlegung eines Servers für einen gesamten WF-Typ) oder es werden unnötige Migrationen erzwungen (bei lokaler Bearbeiterauflösung, wenn der Server bei der Anwendung platziert wird und bei voll verteilten Ansätzen). In beiden Fällen fehlen also jegliche Freiheitsgrade bei der Festlegung ob, wann und wohin migriert werden soll. Diese Freiheitsgrade werden benötigt, um die Kommunikationskosten minimieren zu können. Deshalb entstehen bei den anderen Ansätzen höhere Kommunikationskosten als bei ADEPT. Diese Aussage wurde auch durch die durchgeführten Simulationen bestätigt (siehe Kapitel 9).

ADEPT$_{distribution}$ ist der bislang einzige Ansatz, bei dem die Möglichkeit besteht, eine geeignete Zuordnung der WF-Server zu den Aktivitäten automatisch berechnen zu lassen. Dies ist aber sehr wichtig, weil ein WF-Modellierer mit der Festlegung einer geeigneten Verteilung der Aufgaben auf die WF-Server überfordert wäre. Der Versuch, eine solche manuell festzulegen, würde deshalb höchstwahrscheinlich zu ungünstigen Serverzuordnungen, und damit zu einem schlechten Kommunikationsverhalten führen. Um eine automatische Berechnung von Serverzuordnungen zu ermöglichen, mussten nicht nur die in Abschnitt 4.5 vorgestellten Verteilungsalgorithmen entwickelt werden, sondern auch ein Kostenmodell, mit dem die bei der WF-Ausführung entstehenden Belastungen abgeschätzt

werden können. Für dieses war es wiederum erforderlich, Verfahren zur Berechnung von mehreren Wahrscheinlichkeitsverteilungen zu entwickeln.

Dass der von uns verfolgte Ansatz große Vorteile mit sich bringt, konnte in Abschnitt 9.5 durch die Simulation eines Prozesses gezeigt werden, in dem keine abhängigen Bearbeiterzuordnungen verwendet werden. Obwohl dann keine variablen Serverzuordnungen eingesetzt werden können, zeigt ADEPT im Vergleich zu allen anderen Verteilungsmodellen das mit Abstand beste Kommunikationsverhalten. So ist die Belastung der Gateways, und damit die Kosten für ggf. entfernte Kommunikationen, bei allen anderen Ansätzen annähernd doppelt so groß. Bei einigen Verteilungsmodellen beträgt die Belastung sogar mehr als das Dreifache.

10.2 Variable Serverzuordnungen

Die Klassifikation der verteilten WfMS aus Abschnitt 9.1 zeigt deutlich, dass variable Serverzuordnungen bisher ausschließlich von ADEPT verwendet werden. Dieses Konzept wurde im Rahmen der vorliegenden Arbeit entwickelt, und es gibt auch keine vergleichbaren Ansätze. Dasselbe gilt deshalb natürlich auch für die Verfahren zur automatischen Berechnung von geeigneten variablen Serverzuordnungen.

In der WF-Literatur wurden bisher die Auswirkungen der in der Praxis äußerst relevanten abhängigen Bearbeiterzuordnungen auf eine verteilte WF-Ausführung nicht betrachtet. Bei einigen der in der Literatur beschriebenen Ansätze können abhängige Bearbeiterzuordnungen sogar überhaupt nicht realisiert werden, weil keine globale Bearbeiterauflösung vorgesehen ist, d.h. alle potentiellen Bearbeiter einer Aktivität müssen demselben Domain angehören. Abhängige Bearbeiterzuordnungen machen aber nur dann Sinn, wenn es eine Art „Pivot-Aktivität“ gibt, welche die OE der nachfolgenden (abhängigen) Aktivitäten determiniert (z.B. die betroffene Station innerhalb einer Klinik). Diese Pivot-Aktivität kann dann in den abhängigen Bearbeiterzuordnungen referenziert werden. Sie kann aber nicht realisiert werden, wenn keine globale Bearbeiterauflösung möglich ist, weil ihre potentiellen Bearbeiter unterschiedlichen OE, und damit unterschiedlichen Domains, angehören. Der tatsächlich verwendete Domain steht erst nach Ausführung der Pivot-Aktivität fest. Beispiele für Ansätze mit ausschließlich lokaler Bearbeiterauflösung sind MENTOR [WWWK96b, MWW+98] und WIDE [CGP+96, CGS97].

Ein akzeptables Verhalten bei der Ausführung von Prozessen mit abhängigen Bearbeiterzuordnungen kann, außer durch variable Serverzuordnungen, lediglich bei den in Abschnitt 9.1.4 vorgestellten voll verteilten Ansätzen erreicht werden. Ein solcher Ansatz wird z.B. von INCAS [BMR94, BMR96] und Exotica/FMQM [AMG+95] verfolgt. Hier wird eine WF-Instanz stets zum Bearbeiter der jeweils aktuellen Aktivität migriert, so dass sie sich damit im „richtigen“ Domain befindet. Allerdings führt dieser Ansatz, wie in Kapitel 9 schon ausführlich diskutiert wurde, zu sehr vielen unnötigen Migrationen, die wiederum das Kommunikationsverhalten verschlechtern. Außerdem lässt sich ein voll verteiltes WfMS nur dann sinnvoll einsetzen, wenn zumindest dem Großteil der Aktivitäten exakt ein Bearbeiter zugeordnet ist. Andernfalls wird der Aufwand für die Bearbeiterauflösung zu groß. Aus diesen Gründen wurde für ADEPT kein solcher Ansatz gewählt.

Dass der Nutzen von variablen Serverzuordnungen sehr groß ist, konnte durch die in Abschnitt 9.6 beschriebene Simulation eines Szenarios, in dem mehrere abhängige Bearbeiterzuordnungen auftreten, belegt werden. Hier wird die durchschnittliche Belastung der Teilnetze durch den Einsatz von variablen Serverzuordnungen annähernd halbiert, und zwar nicht nur gegenüber einem zentralen WfMS,

sondern auch im Vergleich zu allen anderen Verteilungsmodellen. Noch eindrucksvoller ist die durch die Verwendung von variablen Serverzuordnungen resultierende geringe Belastung der Gateways, die bei allen anderen Ansätzen sogar mehr als 14 Mal größer ist.

10.3 Optimierung von Migrationen

Wenden wir uns nun der Fragestellung zu, welche Daten einer WF-Instanz bei einer Migration übertragen werden sollen. Wie in Kapitel 6 beschrieben wurde, wird die zu übertragende Datenmenge bei ADEPT$_{distribution}$ derart reduziert, dass eine redundante Datenübertragung ausgeschlossen ist. Dies gilt auch dann, wenn ein WF-Server mehrfach (mit Unterbrechungen) an der Steuerung einer WF-Instanz beteiligt ist, oder wenn die von ihm kontrollierte Aktivität mehrere Vorgängeraktivitäten hat, deren Server Informationen an ihn übermitteln. Bei einigen der in der Literatur beschriebenen verteilten WfMS werden die zu einer WF-Instanz vorhandenen Laufzeitdaten bei einer Migration vollständig übertragen. So wird bei CodAlf [SM96, SM97] und BPAFrame [MMSL96, SM96] eine WF-Instanz als Objekt repräsentiert (inkl. internem Zustand, Datenelementen und der Definition des zugehörigen WF-Typs), welches zwischen den WF-Servern migriert. Auch bei INCAS [BMR94, BMR96] wurde ein solcher Objektmigrationsansatz gewählt, allerdings erfolgt die WF-Definition hier durch Regeln. Da zusätzlich zu den schon genannten Daten auch noch die alten Versionen der Datenelemente (vgl. Datenhistorie aus Abschnitt 3.3.4) im migrierten Objekt enthalten sind, muss bei jeder Migration eine sehr große Datenmenge übertragen werden, was zu einem extrem hohen Kommunikationsaufkommen führen kann. Es gibt aber auch Ansätze, die nicht nur bei Migrationen Daten übertragen: Bei Exotica/FMQM [AMG+95] werden die von einer Aktivitäteninstanz geschriebenen Datenelemente an diejenigen Systemkomponenten versendet, die Aktivitäteninstanzen kontrollieren, welche diese Datenelemente lesen. Die verteilte WF-Ausführung in MENTOR [WWWK96b, WWK+97] basiert auf der Partitionierung von State- und Activitycharts. Um eine zum zentralen Fall äquivalente verteilte WF-Ausführung garantieren zu können, müssen, nach der Beendigung von parallel ausgeführten Aktivitäten, Synchronisationsnachrichten und Datenelemente ausgetauscht werden. Da dies einen großen Kommunikationsaufwand erfordert, werden in [MWW+98] Verfahren vorgestellt, mit denen dieser Aufwand reduziert werden kann. Es wird also nicht das Ziel verfolgt, den Kommunikationsaufwand bei Migrationen zu reduzieren, sondern der Aufwand für die Synchronisation der WF-Server bei parallel ausgeführten Aktivitäten wird begrenzt. Eine solche Synchronisation der WF-Abarbeitung ist in ADEPT$_{distribution}$ nicht notwendig, da die Bearbeitung paralleler Zweige unabhängig voneinander fortschreiten kann.

Einige Ansätze verwenden keine Partitionierung und Migrationen im eigentlichen Sinn, sondern starten Subprozesse auf einem entfernten WF-Server. Dies hat außer den im Folgenden diskutierten Aspekten gewisse Nachteile, die schon in Abschnitt 3.4.2 beschrieben wurden. Bei einer Erweiterung von MOBILE [NSS98, Sch97b, SNS99] wird zur Ausführungszeit der WF-Instanzen entschieden, welcher WF-Server einen Sub-WF kontrollieren soll. Auch WIDE [CGP+96, CGS97] erreicht Verteilung durch die entfernte Ausführung von Subprozessen. Diese Vorgehensweise hat den Vorteil, dass an einen Sub-WF nur die von ihm potentiell benötigten Daten übergeben werden müssen, anstatt dass alle Daten der WF-Instanz migriert werden. Allerdings wird dieser Vorteil durch die in Kapitel 6 vorgestellten Optimierungen mehr als ausgeglichen. Es müssen nämlich evtl. auch Datenelemente an einen Sub-WF übergeben werden, die von diesem überhaupt nicht benötigt werden, z.B. weil sie in einer bei dieser WF-Instanz nicht gewählten bedingten Verzweigung gelesen werden würden. Auch Daten, die in dem Sub-WF nicht direkt verwendet werden, sondern mit denen nur ein weiterer

Sub-WF aufgerufen wird, müssen übertragen werden. Datenelemente können also keine „Partitionen überspringen“. Alle diese Problemfälle können bei ADEPT$_{distribution}$ nicht auftreten, da, wie in Kapitel 6 erläutert wurde, eine redundante Übertragung von Zustandsinformation oder Datenelementen stets ausgeschlossen ist.

Bei WIDE [CGP+96] werden Datenelemente nicht direkt an andere WF-Server übergeben, sondern es werden nur Referenzen auf diese Daten übertragen. Diese Vorgehensweise ist häufig bei solchen Systemen zu finden, die wie WIDE auf einer objektorientierten Systeminfrastruktur (z.B. CORBA) basieren, da diese den ortstransparenten Zugriff auf die eigentlichen Datenelemente ermöglicht. Bei METEOR$_2$ [DKM+97, MPS+98] steuert ein sogenannter „Taskmanager“ die Ausführung eines bestimmten Aktivitätentyps und signalisiert den Taskmanagern der Nachfolgeraktivitäten deren Beendigung. Zugriffe auf Datenelemente erfolgen ortstransparent durch CORBA. Dasselbe gilt für die Systeme METUFlow [GAC+97], MOKASSIN [GJS+99], und WASA$_2$ [Wes99c], bei denen eine Aktivitäteninstanz jeweils durch ein CORBA-Objekt repräsentiert wird. Die Nachteile, die bei all diesen Ansätzen dadurch entstehen, dass bei der Ausführung jeder einzelnen Aktivitäteninstanz entfernt auf diese Daten zugegriffen werden muss, wurden schon ausführlich in Abschnitt 2.3.1 diskutiert.

Zusammenfassend lässt sich feststellen, dass in der Literatur zahlreiche Ansätze für verteiltes WF-Management vorgestellt werden, bei denen eine einzelne WF-Instanz von verschiedenen Servern kontrolliert werden kann. Allerdings werden bei keinem von diesen die bei Migrationen anfallenden Kommunikationskosten – wie durch die in Kapitel 6 vorgestellten Verfahren – minimiert. Die Ansätze ignorieren weitgehend die in einem verteilten WfMS anfallenden hohen Kommunikationskosten. Manche der Ansätze treffen aber Entwurfsentscheidungen, die Einfluss auf die entstehenden Kommunikationskosten haben: (1) Bei Migrationen werden nur Referenzen auf Datenelemente übergeben. Dies führt zu den schon diskutierten Nachteilen. (2) Bei Migrationen wird keine explizite Zustandsinformation übergeben (es werden lediglich die Nachfolgeraktivitäten über die Beendigung der Aktivitäteninstanz informiert). Dies führt dazu, dass keine Information über beendete Aktivitäten verfügbar ist, was die Realisierung fortschrittlicher WF-Konzepte, wie abhängige Bearbeiterzuordnungen und die Überwachung von Zeitbedingungen, beeinträchtigt.

Die in Kapitel 6 vorgestellten Verfahren sind also nicht nur originär in dem Sinne, dass Verfahren dieser „Bauart“ bei keinem anderen Ansatz für verteiltes WF-Management zu finden sind; bei anderen Ansätzen wird nicht einmal ernsthaft das Ziel verfolgt, die bei Migrationen zu übertragende Datenmenge zu reduzieren. Deshalb ist die gesamte Vorgehensweise neuartig im Bereich der WF-Forschung.

10.4 Replikation von Workflow-Servern

Im Kapitel 7 haben wir untersucht, wie eine mögliche Überlastung von WF-Servern geeignet behandelt werden kann. Zu diesem Zweck werden WF-Server repliziert, so dass sich mehrere Server innerhalb desselben Domains befinden können. Um für eine konkrete WF-Instanz einen dieser Server auszuwählen, wurde ein auf Pseudo-Zufallszahlen basierendes Verfahren vorgestellt. Die Besonderheit dieses Verfahrens ist, dass es eine beliebige, vorzugebende Verteilung der Last ermöglicht und dabei keinen zusätzlichen Kommunikationsaufwand verursacht.

Unseres Wissens gibt es bisher keine Arbeiten, die sich mit dem Problem der Überlastung des WF-Servers eines Domains befassen. Bei allen Ansätzen außer ADEPT wird entweder völlig auf die Festlegung einer günstigen Verteilung verzichtet, so dass der Domain des zuständigen WF-Servers

ohnehin irrelevant ist (z.B. bei der zufälligen Serverwahl von Exotica/Cluster [AKA+94]), oder die Überlastungsproblematik wird einfach ignoriert. Besonders kritisch ist Letzteres bei Verteilungsmodellen, bei denen die Bearbeiterauflösung nur lokal im Domain des WF-Servers stattfindet (z.B. MENTOR [WWWK96b, MWW+98] und WIDE [CGP+96, CGS97]). Bei diesen ist es im Falle einer Serverüberlastung nicht möglich, den Domain aufzuspalten oder Aufgaben einem anderen (ungünstigeren) WF-Server zuzuordnen, da sich dadurch nicht mehr alle potentiellen Bearbeiter einer Aktivität im Domain des Servers befinden würden. Eine vorliegende Serverüberlastung kann deshalb nicht behoben werden. Da ADEPT$_{distribution}$ bislang der einzige Ansatz ist, der das in Kapitel 7 adressierte Problem behandelt, wird im Folgenden hauptsächlich diskutiert, wie das vorgestellte Verfahren bei verschiedenen Verteilungsstrategien eingesetzt werden kann.

Einige Forschungsprototypen, die sich nicht primär um Skalierbarkeit kümmern (z.B. Panta Rhei [EG96, EL96a, EL96b], WASA [VWW96b, WHKS98]), und die meisten kommerziellen Systeme verwenden einen zentralen WF-Server. Da dieser einen potentiellen Flaschenhals darstellt, können solche WfMS nicht als skalierbar bezeichnet werden. Bei voll verteilten Ansätzen (z.B. Exotica/FMQM [AMG+95] und INCAS [BMR94, BMR96]) übernimmt der Rechner jedes Benutzers Serverfunktionalität. Da es keine WF-Server im eigentlichen Sinn gibt, bei denen sich die Last konzentriert, können auch keine WF-Server überlastet werden. Deshalb besteht keine Notwendigkeit für die Verwendung der in Kapitel 7 vorgestellten Verfahren. Im Exotica-Projekt wurde ein Ansatz entwickelt, bei dem der WF-Server (Cluster) für eine WF-Instanz bei deren Start zufällig ausgewählt wird [AKA+94]. In diesem Cluster verbleibt der WF für seine gesamte Laufzeit, es gibt also keine Migrationen. Dies entspricht einem zentralen Ansatz mit repliziertem WF-Server. Die Serverwahl findet rein zufällig statt. Da eine WF-Instanz den Cluster nie wechselt, treten die in Abschnitt 7.3.3 beschriebenen Probleme bei der Zusammenführung paralleler Zweige hier nicht auf, so dass die zufällige Serverwahl ausreichend ist.

Bei zahlreichen Ansätzen werden wie bei ADEPT mehrere WF-Server verwendet, wobei ebenfalls ein Wechsel des Servers möglich ist. In MENTOR [WWWK96b, WWK+97, MWW+98] und WIDE [CGP+96, CGS97] wird der WF-Server nahe bei den potentiellen Bearbeitern der Aktivität allokiert; bei METEOR$_2$ [DKM+97, MPS+98], CodAlf [SM96, SM97] und BPAFrame [MMSL96, SM96] liegt er nahe bei der zur Aktivität gehörenden Anwendung. Bei all diesen Ansätzen kann ein WF-Server überlastet sein. Mit dem vorgestellten pseudo-zufälligen Verfahren wäre es möglich, ihn zu replizieren und die Last auf die Replikate zu verteilen.

In MOBILE [HS96, Jab97] werden verschiedene WF-Typen von unterschiedlichen WF-Servern kontrolliert. Migrationen sind nicht vorgesehen. Allerdings können Subprozesse von einem anderen WF-Server kontrolliert werden [NSS98, Sch97b, SNS99]. Dieser wird zur Laufzeit aufgrund verschiedener Kriterien (z.B. Rechte, Gewichte) ausgewählt. Auch bei diesem Ansatz können WF-Server überlastet sein, so dass die vorgestellten Verfahren angewendet werden können. Aufgrund der „Subprozess-Semantik“ werden parallele Zweige stets auf demjenigen WF-Server zusammengeführt, auf dem sie sich aufgeteilt haben. Da deshalb das in Abschnitt 7.3.3 geschilderte Problem beim Zusammenführen paralleler Zweige nicht auftreten kann, ist hier das Verfahren mit zufälliger Serverwahl völlig ausreichend.

Die Anwendung des in Kapitel 7 vorgestellten Verfahrens ist also nicht auf ADEPT beschränkt. Zahlreiche andere Ansätze können auch von ihm profitieren. Dabei genügt für Ansätze ohne Migrationen eine zufällige Serverwahl; ansonsten ist (wie bei ADEPT$_{distribution}$) die pseudo-zufällige Variante am besten geeignet. Bisher ist aber noch bei keinem aus der Literatur bekannten Ansatz ein solches Verfahren vorgesehen.

10.5 Adaptives und verteiltes Workflow-Management

In der WF-Literatur finden sich zahlreiche Arbeiten, die sich mit Skalierbarkeitsfragestellungen und verteilter WF-Ausführung beschäftigen (vgl. Abschnitt 9.1). Ebenso gibt es viele Veröffentlichungen zu dem Thema dynamische WF-Änderungen, die z.B. in [Rei00] diskutiert werden. Jedoch gibt es kaum Projekte, die beide Aspekte gemeinsam betrachten – insbesondere wird deren Zusammenspiel nicht hinreichend gewürdigt. Es ist nicht das Ziel dieser Arbeiten, ein bzgl. der Kommunikationskosten effizientes WfMS zu entwickeln, das skalierbar und flexibel ist. Dieser Aspekt wird in der vorliegenden Arbeit erstmalig systematisch untersucht (vgl. Kapitel 8, [BRD01a]), indem betrachtet wird, wie dynamische Änderungen in einem verteilten WfMS durchgeführt werden sollten, und wie bereits veränderte WF-Instanzen effizient verteilt gesteuert werden können.

WIDE erlaubt dynamische Änderungen einer WF-Vorlage und deren Propagierung auf laufende WF-Instanzen [CCPP98]. Außerdem werden WF-Instanzen verteilt gesteuert [CGP+96], wobei die Bearbeiter einer Aktivität den WF-Server determinieren, der diese Aktivität kontrolliert. Bei MOKASSIN [GJS+99, JH98] und WASA [Wes98, Wes99c] wird die verteilte WF-Ausführung durch eine zugrunde liegende CORBA-Infrastruktur realisiert. Außerdem sind Änderungen auf Schema- und auf Instanzebene möglich, wobei auch Konsistenzfragestellungen betrachtet werden. INCAS [BMR94, BMR96] verwirklicht die Steuerung der WF-Instanzen durch Regeln, die modifiziert werden können, um dynamische Änderungen durchzuführen. Die WF-Steuerung findet verteilt statt, wobei eine WF-Instanz stets von dem Rechner desjenigen Benutzers kontrolliert wird, der die aktuelle Aktivität bearbeitet. Bei all diesen Ansätzen wird aber nicht explizit auf das Zusammenspiel der dynamischen Änderungen und der verteilten WF-Ausführung eingegangen. Diese Fragestellung wurde in ADEPT erstmalig untersucht.

In der Literatur finden sich auch einige Ansätze für verteiltes WF-Management, bei denen eine WF-Instanz während ihrer gesamten Lebenszeit von nur einem einzigen WF-Server kontrolliert wird (z.B. Exotica/Cluster [AKA+94], MOBILE [HS96, Jab97] - wurde aber in [NSS98, Sch97b, SNS99] erweitert). Es finden also keine Migrationen statt, unterschiedliche WF-Instanzen können aber von verschiedenen Servern kontrolliert werden. Da für jede WF-Instanz eine zentrale Kontrollinstanz existiert, könnten dynamische Änderungen bei diesen Ansätzen also genauso wie in einem zentralen WfMS durchgeführt werden. Allerdings ist es bei diesem Verteilungsmodell nicht möglich, für jede einzelne Aktivität einen bzgl. der Kommunikationskosten günstigen WF-Server auszuwählen. Da deshalb bei der „normalen WF-Ausführung“ höhere Mehrkosten entstehen, als die bei den (verhältnismäßig seltenen) dynamischen Änderungen erzielten Einsparungen, haben wir für ADEPT$_{distribution}$ keinen solchen Ansatz gewählt.

Kapitel 11

Stand der Realisierung

Einige der in dieser Arbeit vorgestellten Verfahren wurden prototypisch implementiert, um ihre Umsetzbarkeit zu zeigen, und um ihr Zusammenspiel mit anderen Komponenten zu studieren. Außerdem konnte so die Effektivität der Verfahren gezeigt werden. Im dem vorliegenden Kapitel wird aufgezeigt, wie weit die Implementierung inzwischen fortgeschritten ist. Zu diesem Zweck wird auf die Realisierung einiger bereits umgesetzter Konzepte jeweils kurz eingegangen.

Im Abschnitt 11.1 wird beschrieben, wie ein auf statischen Serverzuordnungen basierendes verteiltes WfMS realisiert werden kann, und wie die benötigten geeigneten Serverzuordnungen berechnet werden können. Dies erfolgte auf Basis eines kommerziellen (zentralen) WfMS. Der Abschnitt 11.2 beschäftigt sich mit der Ausführung von Prozessen, bei denen variable Serverzuordnungen verwendet werden. Die entsprechenden Verfahren wurden für unseren Prototypen ADEPT$_{workflow}$ realisiert. Im Abschnitt 11.3 wird aufgezeigt, in wie weit die in Kapitel 6 beschriebenen Optimierungen der bei Migrationen zu übertragenden Datenmenge umgesetzt wurden. Der Abschnitt 11.4 betrachtet schließlich noch die Implementierung von dynamischen Ablaufänderungen in einem verteilten WfMS. Auch dieses Kapitel schließt mit einer Zusammenfassung.

11.1 Statische Partitionierung und Migration

Das Konzept der statischen Serverzuordnungen und deren automatische Berechnung wurden schon zu einem frühen Zeitpunkt dieser Arbeit entwickelt [BD97]. Um die verwendeten Kostenformeln und die Verteilungsalgorithmen verifizieren zu können, wurden diese implementiert. Außerdem sollte durch Messungen belegt werden, dass die vorgestellten Verfahren tatsächlich dazu geeignet sind, die in den Teilnetzen des Kommunikationsnetzwerks entstehende Last zu reduzieren. Deshalb war auch eine Realisierung der verteilten Ausführung von WF-Instanzen erforderlich. Dies alles erfolgte auf Basis eines kommerziellen (zentralen) WfMS [End98].

Die statisch definierte verteilte WF-Ausführung wurde auf Basis von IBM FlowMark [IBM96b] umgesetzt. Der Hauptgrund für die Wahl dieses Produktes war, dass es bei FlowMark möglich ist, WF-Vorlagen in Form von FlowMark-Definition-Language (FDL) -Dateien zu exportieren und zu importieren. Dies eröffnet die Möglichkeit zur Manipulation dieser Vorlagen: Die Modellierungskomponente von FlowMark kann verwendet werden, um FDL-Definitionen der entsprechenden WF-Vorlagen zu erzeugen. Für die Berechnung von geeigneten Serverzuordnungen zusätzlich benötigte statistische Information (vgl. Abschnitt 4.2) kann in Kommentarfeldern abgelegt werden, welche

ebenfalls in die FDL-Dateien übernommen werden. Der Verteilungsalgorithmus kann nun realisiert werden, indem die FDL-Definition eingelesen wird, geeignete Serverzuordnungen berechnet werden und getrennte FDL-Definitionen für die verschiedenen WF-Server erzeugt werden. Mit dieser Vorgehensweise wurde die in Abschnitt 4.5.2 vorgestellte Variante des Verteilungsalgorithmus mit polynomiellem Laufzeitverhalten implementiert, bei der initial berechnete Partitionen ggf. zusammengefasst werden. Die einzelnen vom Verteilungsalgorithmus erzeugten FDL-Dateien werden von den entsprechenden WF-Servern importiert, so dass die Server über die für sie jeweils relevanten Partitionen des Gesamtprozesses verfügen. Da in einer FDL-Datei auch das zugehörige Organisationsmodell enthalten ist, war es außerdem möglich, den Algorithmus aus Abschnitt 4.6 umzusetzen, um eine geeignete Verteilung der Benutzer auf die Domains zu ermitteln.

Die Fähigkeit von FlowMark zur entfernten Ausführung von Subprozessen [IBM96b] konnte leider nicht zur Realisierung der verteilten WF-Steuerung genutzt werden, weil die Semantik nicht derjenigen von ADEPT$_{distribution}$ entspricht. Da bei FlowMark die WF-Kontrolle nach Beendigung eines Sub-WF stets zum Server des Super-WF zurückkehrt, und keine Migrationen im eigentlichen Sinne durchgeführt werden, wäre ADEPT nicht sauber nachgebildet worden, was die Ergebnisse verfälscht hätte. Deshalb war es notwendig, die Migrationen selbst zu realisieren. Zu diesem Zweck fügt der Verteilungsalgorithmus am Anfang und Ende jeder Partition einige zusätzliche (automatisch ausgeführte) Aktivitäten ein. Dies ist beispielhaft für die Sequenz der Aktivitäten a und b aus Abb. 11.1a in Abb. 11.1b dargestellt. Am Ende einer Partition wird die Aktivität $m1$ eingefügt, welche die Laufzeitdaten der WF-Instanz an eine Aktivität $m2$ sendet. Diese wiederum wurde am Anfang der Zielpartition eingefügt und hat die Aufgabe, die empfangenen Instanzdaten in der WF-DB zu speichern. Diese beiden Aktivitäten führen also die eigentlich Migration durch. Allerdings sind sie noch nicht ausreichend, um in FlowMark eine verteilte WF-Ausführung zu realisieren. Wird nämlich aufgrund einer Dead-Path-Elimination[1] die Aktivität a nicht ausgeführt, so wird auch die Aktivität $m1$ nicht ausgeführt, weshalb die Migration zu $m2$ niemals durchgeführt wird. Dadurch wäre die Zielpartition blockiert, weil FlowMark den Zustand einer Aktivität erst dann evaluiert, wenn alle Eingangskanten (mit True oder False) signalisiert wurden. Um eine solche Blockadesituation zu vermeiden, wird als zusätzlicher Vorgänger von $m1$ die Aktivität $dummy$ eingefügt, die selbst keine eingehenden Kanten hat, weshalb sie beim Start der WF-Instanz sofort ausgeführt wird. Somit evaluiert die Kante $dummy \rightarrow m1$ stets zu True, und die Aktivität $m1$ wird (wegen ihrer Startbedingung „At-least-one“) stets ausgeführt. Dadurch ist die Durchführung der Migration sichergestellt. Allerdings muss dem Zielknoten $m2$ noch mitgeteilt werden, ob die Kante $m2 \rightarrow b$ mit True oder False signalisiert werden soll. Deshalb wird eine speziell für diesen Zweck eingeführte Variable $Edge$ im Falle der Ausführung der Aktivität $eval$ auf True gesetzt und bei der Migration mitübertragen. Am Wert dieser Variablen $Edge$ kann bei der Zielpartition erkannt werden, ob die Aktivität $eval$ ausgeführt wurde oder nicht, und damit, in welchem Modus die Kante $m2 \rightarrow b$ signalisiert werden muss. Dass diese Kante entsprechend der Variablen $Edge$ mit True oder False signalisiert wird, wird durch eine entsprechende Kantenbedingung erreicht (vgl. Abb. 11.1b).

Um die Qualität der berechneten Serverzuordnungen beurteilen zu können, wurden WF-Instanzen mit zentraler und mit verteilter WF-Steuerung ausgeführt. Eine speziell für diesen Zweck entwickelte Komponente simuliert eine größere Anzahl von WF-Clients, so dass tatsächlich eine nennenswerte Last in den einzelnen Teilnetzen entsteht. Diese wurde gemessen. Dabei hat sich gezeigt, dass schon

[1] Eine Dead-Path-Elimination wird in FlowMark z.B. dann durchgeführt, wenn sich die Aktivitäten in einem nicht gewählten Zweig einer bedingten Aufspaltung befinden. In einem solchen Fall werden die entsprechenden Kanten mit False signalisiert. Aktivitäten mit False-signalisierten Eingangskanten werden nicht ausgeführt, weshalb ihre Ausgangskanten ebenfalls mit False signalisiert werden.

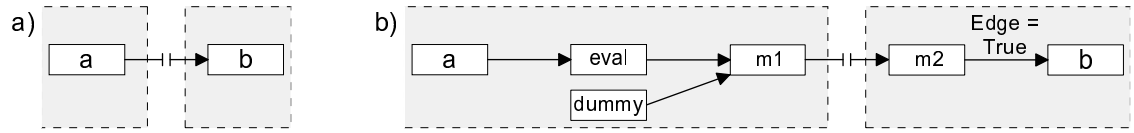


Abbildung 11.1 Realisierung einer Migration auf Basis von FlowMark.

bei Verwendung von nur 3 Domains die durchschnittliche Last der Teilnetze weniger als 50% der Last des Teilnetzes eines zentralen WF-Servers beträgt. Damit könnte belegt werden, dass statisch definierte Serverzuordnungen ein geeignetes Mittel sind, um die Belastung der Teilnetze zu reduzieren.

11.2 Variable Serverzuordnungen

Eine verteilte WF-Ausführung inkl. variabler Serverzuordnungen auf Basis eines kommerziellen WfMS zu realisieren, ist äußerst problematisch, da es hierfür im Gegensatz zu der in Abschnitt 11.1 beschriebenen Umsetzung von statischen Serverzuordnungen nicht ausreicht, Partitionen zu bilden und diese WF-Servern zuzuordnen. Deshalb wurden variable Serverzuordnungen in den WfMS-Prototypen ADEPT$_{workflow}$ integriert [Zei99]. Die in dieser Arbeit beschriebenen Konzepte wurden hierbei direkt umgesetzt, weshalb ihre Implementierung im Folgenden nur sehr verkürzt dargestellt wird.

Die Architektur von ADEPT$_{workflow}$ [HRB$^+$00] wird im Folgenden nicht detailliert beschrieben. Stattdessen wird nur auf Aspekte eingegangen, welche die verteilte WF-Ausführung betreffen. Jeder durch einen WF-Client angestoßene Auftrag wird in die Input-Queue des entsprechenden WF-Servers eingefügt (vgl. Abb. 11.2). Sobald ein Aktivitätsträger (Prozess, Thread) verfügbar ist, wird der Auftrag von diesem bearbeitet. Dabei durchquert jeder Aufruf die Verteilungsschicht. Diese ruft elementare Zugriffsfunktionen auf, um auf die lokal vorhandenen Daten zuzugreifen. Reichen diese zur Bearbeitung einer Aufgabe nicht aus, so wird mit anderen WF-Servern kommuniziert. Dies ist z.B. erforderlich, um eine Migration oder eine verteilte dynamische Änderung durchzuführen.

Die variablen Serverzuordnungsausdrücke selbst sind als Attribute der Aktivitätenknoten realisiert. Diese Ausdrücke werden für zur Aktivierung anstehende Aktivitäten, unter Verwendung der Ablaufhistorie und des Organisationsmodells, ausgewertet, um den tatsächlichen WF-Server der Aktivitäteninstanz zu ermitteln. Allerdings unterstützt die aktuelle Implementierung nur die Typen 1-3 der Serverzuordnungsausdrücke aus Abschnitt 5.1.2, d.h. Funktionen in Serverzuordnungen sind noch nicht realisiert. Diese eingeschränkte Umsetzung ist akzeptabel, da mit ihr gezeigt werden konnte, dass von einem WfMS tatsächlich auch solche Prozesse ausgeführt werden können, bei denen variable Serverzuordnungen verwendet werden. Um dies zu ermöglichen, musste natürlich auch das Protokoll aus Abschnitt 5.2.2 implementiert werden, damit der WF-Server einer Synchronisationsaktivität von allen Vorgängeraktivitäten bestimmt werden kann.

11.3 Optimierung von Migrationen

Die meisten der im Kapitel 6 beschriebenen Verfahren wurde im Prototypen ADEPT$_{workflow}$ implementiert [Zei99]. So wird die Migration der Zustandsinformation realisiert, indem die Ablaufhistorie der WF-Instanz übertragen wird. Diese wird aber nicht bei jeder Migration komplett übertragen, sondern es wurde das in Abschnitt 6.2.2 beschriebene Verfahren zum Anfordern von noch benötigten

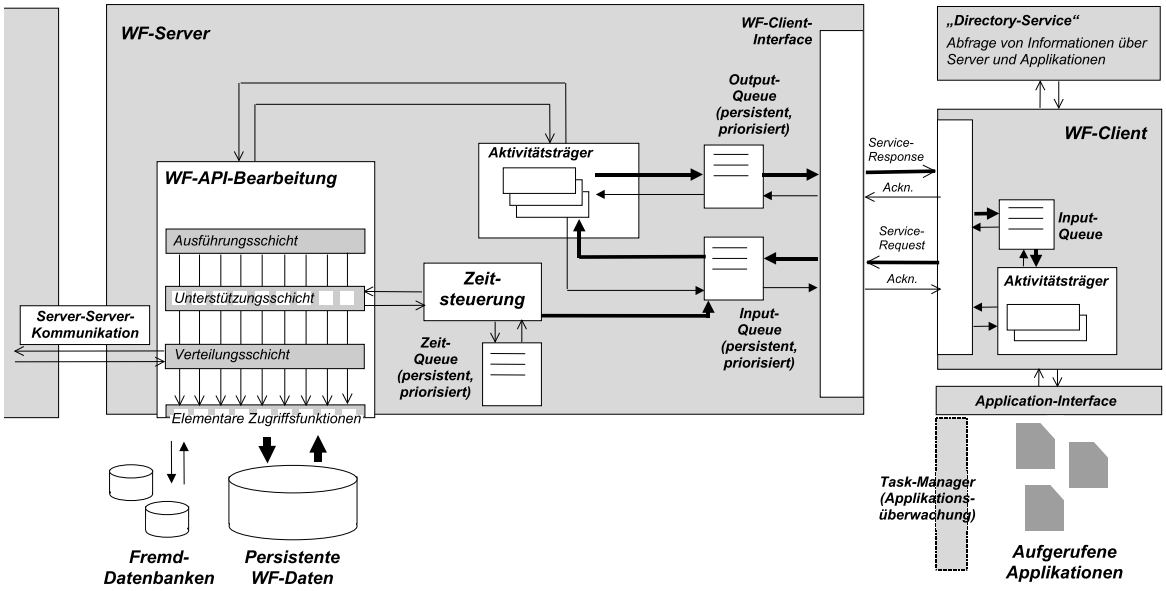


Abbildung 11.2 Grobarchitektur des ADEPT-Workflow-Management-Systems.

Ablaufhistorieneinträgen realisiert. Zur Übertragung von Datenelementen wird das in Abschnitt 6.3.1 beschriebene Verfahren zum Versenden von kleinen Datenelementen verwendet. Bei diesem werden nur diejenigen Datenelemente übertragen, die von Aktivitäten erzeugt wurden, deren Ablaufhistorieneinträge bei der entsprechenden Migration angefordert wurden. Es werden also nicht bei jeder Migration alle Datenelemente der WF-Instanz übertragen, sondern jede Version eines Datenelements wird maximal ein Mal an einen WF-Server transferiert. Ein spezielles Verfahren zum Versenden bzw. Anfordern von besonders großen Datenelementen wurde nicht realisiert, da solche zur Zeit vom ADEPT-Prototypen nicht explizit unterstützt werden.

11.4 Verteilte dynamische Änderungen

Auch das Zusammenspiel von dynamischen Einfüge- und Löschoperationen mit der verteilten WF-Ausführung wurde implementiert [Zei99]. In diesem Zusammenhang ist sogar die Verwendung von variablen Serverzuordnungen möglich.

Um in einem verteilten WfMS dynamische Änderungsoperationen zu ermöglichen, wurden die in Abschnitt 8.3.1 beschriebenen Verfahren zur Synchronisation der aktuell an einer WF-Instanz beteiligten WF-Server und zur Übertragung der Änderungshistorie realisiert. Bei der vorliegenden Implementierung wird ausschließlich der bei einem Migrationszielserver tatsächlich noch benötigte Teil der Änderungshistorie übertragen, um das zu kommunizierende Datenvolumen klein zu halten. Außerdem wurde der bei einer Synchronisation der WF-Server erforderliche Austausch von Zustandsinformation so realisiert, dass (ebenso wie bei Migrationen) nur der jeweils noch nicht bekannte Teil der Ablaufhistorie übertragen wird. Des Weiteren war es zur Realisierung von dynamischen Änderungsoperationen erforderlich, ein Verfahren zu implementieren, mit dem für neu eingefügte Aktivitäten auf effiziente Weise eine Serverzuordnung festgelegt wird. Hierfür wurde das in Abschnitt 8.3.2 vorgestellte Verfahren verwendet, bei dem das Entstehen zusätzlicher Migrationen vermieden wird.

11.5 Zusammenfassung

Durch die Implementierung der meisten der in dieser Arbeit vorgestellten Verfahren konnte gezeigt werden, dass sie „funktionieren“. Der Großteil der Verfahren wurde im Prototypen ADEPT$_{workflow}$ implementiert, der u.a. auf der CeBIT'2000, der EDBT'2000[2] und der BIS'2000[3] vorgestellt wurde. Insbesondere wurden alle zur Ausführungszeit der WF-Instanzen relevanten Algorithmen in diesem Prototypen realisiert, wodurch auch die Korrektheit ihres Zusammenspiels gezeigt werden konnte. So ist es z.B. möglich, dynamische Änderungen in einem WfMS mit variablen Serverzuordnungen und optimierten Migrationen durchzuführen. Des Weiteren wurden gewisse Konzepte auf Basis von IBM FlowMark umgesetzt. Dadurch wurde gezeigt, dass diese prinzipiell in kommerzielle WfMS integriert werden können. Außerdem war es so möglich, durch Messungen an einem realen System, die Effektivität der Verfahren zu belegen.

[2] 7th International Conference on Extending Database Technology, Konstanz, März 2000.
[3] 4th International Conference on Business Information Systems, Posen, April 2000.

Kapitel 12

Zusammenfassung der Ergebnisse

Von der Seite der Anwender von Informationssystemen wird zunehmend gefordert, dass diese prozessorientiert realisiert werden, weil dies die Benutzung des Informationssystems deutlich vereinfacht, und hilft, Fehler bei der Bearbeitung von Geschäftsprozessen zu vermeiden. Wie eingangs dieser Arbeit gezeigt wurde, ist die Realisierung von komplexen prozessorientierten Anwendungssystemen mit konventioneller Implementierungstechnik nicht mit vertretbarem Aufwand möglich. Durch die Verwendung eines WfMS kann jedoch die Umsetzung eines solchen Systems drastisch vereinfacht werden, da der applikationsspezifische Code der einzelnen Anwendungen von der Definition der Ablauflogik des Geschäftsprozesses getrennt wird. Dadurch wird die zu entwickelnde Gesamtapplikation in überschaubare Einheiten aufgeteilt und die Ablaufsteuerung des Geschäftsprozesses wird durch das WfMS realisiert.

Ein Mangel heutiger WfMS ist allerdings ihre unzureichende Skalierbarkeit bei hoher Last. Dies stellt insbesondere für die Realisierung von unternehmensweiten und -übergreifenden Anwendungssystemen ein großes Problem dar, weil bei solchen Systemen eine große Anzahl von Benutzern bedient werden muss, die u.U. teilweise nur über langsame Netzwerke miteinander verbunden sind. Deshalb stellt sich die Frage, wie ein skalierbares WfMS effizient realisiert werden kann, das heißt so, dass die Belastung der WF-Server und des Kommunikationssystems minimiert wird. Dass auch die Belastung des Kommunikationssystems sehr schnell zu einem Problem werden kann, wenn ein WfMS ohne Berücksichtigung dieses Aspekts realisiert wird, wurde in dieser Arbeit mit Hilfe eines Zahlenbeispiels demonstriert.

In der vorliegenden Arbeit wurde gezeigt, wie es durch eine intelligente Verteilungsstrategie möglich ist, ein WfMS so zu realisieren, dass die entstehende Last auf die WF-Server verteilt wird, und außerdem das zu kommunizierende Datenvolumen klein bleibt. Diese Tatsache wurde durch verschiedene Simulationen belegt. Außerdem wurden die vorgestellten Verfahren so gestaltet, dass es auch in dem resultierenden verteilten WfMS möglich ist, fortschrittliche Funktionalitäten, wie die Änderung von WF-Instanzen zu deren Ausführungszeit oder abhängige Bearbeiterzuordnungen, zu realisieren. Im Folgenden fassen wir die wichtigsten Ergebnisse, die somit auch den Kern von ADEPT$_{distribution}$ bilden, nochmals kurz zusammen:

Grundlage des Workflow-Management-Systems

Um die anvisierten Ziele erreichen zu können, war es wichtig, einige grundlegende Entscheidungen zu treffen. So wurden geeignete Verfahren zur Realisierung des Datenflusses und zur Aktualisierung

der Arbeitslisten der Benutzer gewählt. Eine entscheidende Rolle spielt auch die Art und Weise, wie die zugrunde liegende Systeminfrastruktur aufgebaut ist. Deshalb wurden mehrere Möglichkeiten betrachtet und analysiert. Dabei hat sich gezeigt, dass es am günstigsten ist, in den Teilnetzen der Benutzer jeweils mindestens einen WF-Server zu platzieren. Nur durch eine solch hochgradig verteilte Infrastruktur ist es möglich, das Entstehen von Flaschenhälsen bei den WF-Servern und im Kommunikationssystem zu vermeiden.

Verteilungsstrategie

Das zentrale Element dieser Arbeit ist die Strategie, mit der die zu erledigenden Aufgaben auf die WF-Server verteilt werden. Mit Hilfe von Simulationen wurde gezeigt, dass das dafür gewählte Verteilungsmodell äußerst positive Auswirkungen auf die Belastung der WF-Server und des Kommunikationssystems hat. Dieses Verteilungsmodell basiert darauf, den WF-Server einer Aktivität möglichst nahe bei deren potentiellen Bearbeitern zu wählen. Um dies zu ermöglichen werden die WF-Vorlagen zur Modellierungszeit partitioniert, was zu der Notwendigkeit von Migrationen zur Ausführungszeit der WF-Instanzen führt. Außerdem ist eine globale (domainübergreifende) Auflösung der Bearbeiterzuordnungen unbedingt erforderlich, um keine Freiheitsgrade bei der Festlegung des WF-Servers einer Aktivität zu verlieren. Eine Besonderheit von ADEPT$_{distribution}$ ist, dass automatisch geeignete Serverzuordnungen für die einzelnen Aktivitäten einer WF-Vorlage ermittelt werden. Dies geschieht durch Verteilungsalgorithmen, die ein polynomielles Laufzeitverhalten aufweisen. Um diese anwenden zu können, war es erforderlich, ein Kostenmodell zu entwickeln, mit dem die unter Verwendung gegebener Serverzuordnungen entstehenden Kosten abgeschätzt werden können.

Damit eine WF-Instanz verteilt von mehreren WF-Servern gesteuert werden kann, sind Migrationen erforderlich. Um diese realisieren zu können, musste formal festgelegt werden, welche Zustände eine Migration auslösen und welche Aktionen bei einer solchen Migration durchgeführt werden müssen. Damit das Kommunikationsverhalten von Migrationen verbessert wird, wurden außerdem Verfahren entwickelt, mit denen Migrationen besonders effizient durchgeführt werden können. Diese Verfahren basieren darauf, dass Informationen nicht einfach bei Migrationen übertragen werden, sondern dass vom Zielserver noch benötigte Information gezielt angefordert wird. Bei diesen Verfahren wird lediglich ein Teil der Ablaufhistorie der betroffenen WF-Instanz übertragen, um ihren Zustand zu übermitteln. Datenelemente werden bei diesem Verfahren niemals redundant übertragen. Für große Datenelemente wurde ein Verfahren entwickelt, bei dem diese nur dann übertragen werden, wenn sie auch tatsächlich vom Zielserver benötigt werden. In diesem Fall werden die Datenelemente sogar bei dem dafür optimal geeigneten WF-Server angefordert.

Umgang mit abhängigen Bearbeiterzuordnungen

Um viele der in der Praxis vorkommenden Abläufe abbilden zu können, sind Aktivitäten mit abhängigen Bearbeiterzuordnungen erforderlich. Für diese Aktivitäten entsteht allerdings ein Problem bei der Festlegung des optimal geeigneten WF-Servers, da der Domain der Bearbeiter einer solchen Aktivität zur Modellierungszeit noch nicht bekannt ist. Eine statische Festlegung des WF-Servers ist in solchen Fällen ungeeignet. Deshalb wurden die sog. variablen Serverzuordnungen eingeführt. Dass diese eine große Effizienzsteigerung mit sich bringen, wurde durch Simulationen gezeigt. Allerdings führen variable Serverzuordnungen auch zu einigen Schwierigkeiten: So musste ein Protokoll entwickelt werden, mit dem parallele Ausführungszweige synchronisiert werden können, da variable Serverzuordnungen nicht in allen Zweigen aufgelöst werden können. Auch die automatische Berechnung von

geeigneten variablen Serverzuordnungen gestaltet sich kompliziert, da die vom Kostenmodell benötigen Wahrscheinlichkeitsverteilungen jetzt von Vorgängeraktivitäten abhängen. Die entsprechenden Verfahren sind sehr komplex und aufwendig. Dies ist aber akzeptabel, weil sie zur Modellierungszeit ausgeführt werden (weshalb sie die WF-Server nicht belasten) und weiterhin eine polynomielle Laufzeit haben. Außerdem verbessern sie die Effizienz der WF-Steuerung drastisch.

Lastverteilung auf mehrere Server eines Domains

Um die Überlastung von Systemkomponenten zu verhindern, darf natürlich nicht ausschließlich das Kommunikationsverhalten betrachtet werden. Auch WF-Server können einen Engpass darstellen. Deshalb muss es möglich sein, mehrere WF-Server in demselben Domain zu betreiben. Dann stellt sich aber die Frage, wie die anstehenden Aufgaben auf diese verteilt werden sollen. In dieser Arbeit wurde ein sehr trickreiches Verfahren vorgestellt, das keine Kommunikation oder Synchronisation zwischen diesen Servern erfordert. Es hat außerdem die sehr schöne Eigenschaft, dass auf einfache Weise eine beliebige Lastverteilung zwischen diesen Servern eingestellt (und verändert) werden kann. Somit ist die Problematik der Überlastung für alle Komponenten (WF-Server, Teilnetze, Gateways) eines WfMS gelöst.

Unterstützung fortschrittlicher Workflow-Konzepte

Um ein WfMS in der Praxis tatsächlich einsetzen zu können, müssen zahlreiche fortschrittliche WF-Konzepte unterstützt werden. Sämtliche in dieser Arbeit vorgestellten Verfahren sind so gestaltet, dass sie keinem denkbaren WF-Konzept im Wege stehen, weil den aktuell an einer WF-Instanz beteiligten Servern die gesamte Steuerungsinformation (aus den Ablaufhistorien) bekannt ist. Allerdings erfordert die effiziente Umsetzung bestimmter Konzepte in einem verteilten WfMS die Entwicklung besonderer Verfahren. So wurde schon erwähnt, dass zur effizienten Unterstützung von abhängigen Bearbeiterzuordnungen die variablen Serverzuordnungen benötigt werden. Auch für das effiziente Zurücksetzen von WF-Instanzen in ADEPT$_{distribution}$ wurde eine Lösung gefunden: Migrationen, die bei der Vorwärtsausführung stattgefunden haben, werden beim Zurücksetzen in umgekehrter Richtung ausgeführt. Außerdem werden die Historien der Datenelemente durch die ehemals an der WF-Instanz beteiligten Servern aufbewahrt. Dadurch ist es nicht notwendig, die Datenhistorie bei Migrationen zu übertragen, sondern es muss nur der aktuell gültige Wert übermittelt werden.

Die Umsetzung von kontrollierten dynamischen Änderungen einer WF-Instanz stellt in einem verteilten WfMS ein besonders schwieriges Problem dar, weil hierfür eigentlich eine zentrale Kontrollinstanz benötigt wird. Dennoch wurde in dieser Arbeit ein Verfahren entwickelt, das die effiziente verteilte Steuerung von veränderten WF-Instanzen ermöglicht. Dieses basiert darauf, dass bei Migrationen nur Teile der Änderungshistorie übertragen werden und nicht der gesamte neu entstandene WF-Graph. Auch die dynamischen Änderungsoperationen selbst können mit akzeptablem Synchronisationsaufwand realisiert werden, indem nur diejenigen WF-Server einbezogen werden, die aktuell tatsächlich an der Steuerung der WF-Instanz beteiligt sind.

Ausblick

Wie so oft wirft die Bearbeitung eines Problems neue bzw. weiterführende Fragestellungen auf, die Gegenstand von Folgearbeiten sein könnten. Aufbauend auf die vorliegende Arbeit bieten sich unter

anderem die folgenden Themen für weitergehende Untersuchungen an:

- Zusammengesetzte (komplexe) abhängige Bearbeiterzuordnungen sprengen die Möglichkeiten der in dieser Arbeit eingeführten variablen Serverzuordnungen. Wenn solche Bearbeiterzuordnungen tatsächlich benötigt werden, dann ist es evtl. sinnvoll, zusammengesetzte variable Serverzuordnungen zu verwenden. Dieser Aspekt wurde schon in Abschnitt 5.5 diskutiert.
- Es kann Sinn machen, zur Ausführungszeit einer WF-Instanz eine Serverzuordnung zu verändern, um so den Ausfall oder die Überlastung einer Systemkomponente (WF-Server, Teilnetz, Gateway) zu kompensieren. Allerdings wird die entsprechende Aktivität evtl. erst viel später ausgeführt, so dass zu diesem Zeitpunkt eine veränderte Lastsituation vorherrscht. Deshalb sollte nicht nur untersucht werden, wie eine solche dynamische Veränderung einer Serverzuordnung realisiert werden kann, sondern auch, in welchen Situationen diese überhaupt Vorteile bringt.
- In Abschnitt 8.4 wurden einige Verbesserungsmöglichkeiten für die Durchführung von dynamischen Änderungen in einem verteilten WfMS angesprochen: So können noch intelligentere Verfahren zur Festlegung des WF-Servers von neu eingefügten Aktivitäten entwickelt werden, die zusätzlich statistische Information berücksichtigen. Außerdem sind Änderungsoperationen denkbar, in die nur ein Teil der aktuell an der betroffenen WF-Instanz beteiligten WF-Server einbezogen wird.
- Es können intelligente Verfahren entwickelt werden, um das Zusammenspiel der verteilten WF-Ausführung mit anderen WF-Konzepten effizienter zu gestalten. So ist z.B. im Zusammenhang mit der Überwachung von Zeitbedingungen zu prüfen, ob es notwendig ist, beim Start und Ende jeder Aktivitäteninstanz den entsprechenden Zeitpunkt (sofort) allen anderen an der WF-Instanz beteiligten Servern mitzuteilen. Ein solches Vorgehen führt nämlich zu einer großen Anzahl von zu übertragenden Nachrichten. Um dieses Problem zu lösen, werden Verfahren benötigt, die entscheiden, ob ein bestimmter Start- bzw. Endezeitpunkt für die anderen WF-Server überhaupt relevant ist.
- Um den Ausfall von WF-Servern zu kompensieren, werden üblicherweise Backup-Server [KAGM96] eingesetzt. Da in diesen Servern WF-Daten repliziert gespeichert werden, stellen sie auch eine Art von verteilter WF-Ausführung dar. Allerdings ist das Ziel der Verwendung von solchen Stand-By-Systemen die Erhöhung der Verfügbarkeit – sie verschlechtern das Kommunikationsverhalten sogar. Deshalb stellt sich die Frage nach Verfahren, mit denen die effiziente verteilte WF-Ausführung und die Erhöhung der Verfügbarkeit verbunden werden können, ohne dass das Kommunikationsverhalten (deutlich) beeinträchtigt wird.

Einige der angesprochenen Themen sind schon oder werden gerade (zumindest teilweise) im ADEPT-Projekt bearbeitet. Allerdings kann man keine der Fragestellungen schon als gelöst bezeichnen. Es bleibt also noch viel zu tun, bis WfMS hoffentlich so selbstverständlich zur Anwendungsentwicklung eingesetzt werden, wie heute DBMS. Es ist zu hoffen, dass die Ergebnisse der vorliegenden Arbeit einen Beitrag geleistet haben, um dieses Ziel zu erreichen.

Literaturverzeichnis

[AAE+96] G. Alonso, D. Agrawal, A. El Abbadi, M. Kamath, R. Günthör und C. Mohan: *Advanced Transaction Models in Workflow Contexts*. In: *Proc. 12th Int. Conf. on Data Engineering*, Februar 1996.

[AAHD97] B. Arpinar, S. Arpinar, U. Halici und A. Dogac: *Correctness of Workflows in the Presence of Concurrency*. In: *Proc. Next Generation Information Technologies and Systems Conf.*, Israel, Juni 1997.

[Ack90] D.H. Ackley: *An Empirical Study of Bit Vector Function Optimization.* In: *Genetic Algorithms and Simulated Annealing.* Morgan Kaufmann, 1990.

[ACM94] J.M. Andrade, M.T. Carges und M.R. MacBlane: *The TUXEDO System: An Open Online Transaction Processing Environment.* IEEE Computer Society, Bulletin of the Technical Committee on Data Engineering, 17(1):34–39, März 1994.

[ACPT99] P. Atzeni, S. Ceri, S. Paraboschi und R. Torlone: *Database Systems: Concepts, Languages and Architectures*. McGraw-Hill, 1999.

[AGK+96] G. Alonso, R. Günthör, M. Kamath, D. Agrawal, A. El Abbadi und C. Mohan: *Exotica/FMDC: A Workflow Management System for Mobile and Disconnected Clients.* Distributed and Parallel Databases, 4(3):229–247, Juli 1996.

[AKA+94] G. Alonso, M. Kamath, D. Agrawal, A. El Abbadi, R. Günthör und C. Mohan: *Failure Handling in Large Scale Workflow Management Systems*. Technischer Bericht RJ9913, IBM Almaden Research Center, November 1994.

[Amd67] G. Amdahl: *Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities.* In: *Proc. 30th AFIPS Spring Joint Computing Conf.*, S. 483–485, Washington, DC, 1967.

[AMG+95] G. Alonso, C. Mohan, R. Günthör, D. Agrawal, A. El Abbadi und M. Kamath: *Exotica/FMQM: A Persistent Message-Based Architecture for Distributed Workflow Management.* In: *Proc. IFIP Working Conf. on Information Systems for Decentralized Organisations*, Trondheim, August 1995.

[BB93] G. Brassard und P. Bratley: *Algorithmik: Theorie und Praxis*. Wolfram, 1993.

[BD97] T. Bauer und P. Dadam: *A Distributed Execution Environment for Large-Scale Workflow Management Systems with Subnets and Server Migration.* In: *Proc. 2nd IFCIS Conf. on Cooperative Information Systems*, S. 99–108, Kiawah Island, SC, Juni 1997.

[BD98] T. Bauer und P. Dadam: *Architekturen für skalierbare Workflow-Management-Systeme – Klassifikation und Analyse.* Ulmer Informatik-Berichte 98-02, Universität Ulm, Fakultät für Informatik, Januar 1998.

[BD99a] T. Bauer und P. Dadam: *Efficient Distributed Control of Enterprise-Wide and Cross-Enterprise Workflows.* In: *Proc. Workshop Enterprise-wide and Cross-enterprise Workflow Management: Concepts, Systems, Applications, 29. Jahrestagung der GI*, S. 25–32, Paderborn, Oktober 1999.

[BD99b] T. Bauer und P. Dadam: *Verteilungsmodelle für Workflow-Management-Systeme – Klassifikation und Simulation.* Informatik Forschung und Entwicklung, 14(4):203–217, Dezember 1999.

[BD00a] T. Bauer und P. Dadam: *Efficient Distributed Workflow Management Based on Variable Server Assignments.* In: *Proc. 12th Conf. on Advanced Information Systems Engineering*, S. 94–109, Stockholm, Juni 2000.

[BD00b] T. Bauer und P. Dadam: *Variable Serverzuordnungen und komplexe Bearbeiterzuordnungen im Workflow-Management-System ADEPT*. Ulmer Informatik-Berichte 2000-02, Universität Ulm, Fakultät für Informatik, Februar 2000.

[Bem92] T. Bemmerl: *Programmierung skalierbarer Multiprozessoren*. BI-Wissenschaftsverlag, 1992.

[BF99] E. Bertina und E. Ferrari: *The Specification and Enforcement of Authorization Constraints in Workflow Management Systems.* ACM Transactions on Information System Security, 2(1):65–104, Februar 1999.

[BHJ+96] C. Bußler, P. Heinl, S. Jablonski, H. Schuster und K. Stein: *Architektur von unternehmensweit einsetzbaren Workflow-Management–Systemen*. In: *Proc. MobIS 96, Rundbrief des GI-Fachausschusses 5.2*, S. 73–77, Oktober 1996.

[BM93] M. Batubara und A.J. McGregor: *An Introduction to B-ISDN and ATM*. Technischer Bericht TR 93/14, Monash University, Clayton, Australia, September 1993.

[BMR94] D. Barbará, S. Mehrotra und M. Rusinkiewicz: *INCAS: A Computational Model for Dynamic Workflows in Autonomous Distributed Environments.* Technischer Bericht, Matsushita Information Technology Laboratory, Princeton, NJ, Mai 1994.

[BMR96] D. Barbará, S. Mehrotra und M. Rusinkiewicz: *INCAs: Managing Dynamic Workflows in Distributed Environments.* Journal of Database Management, Special Issue on Multidatabases, 7(1):5–15, 1996.

[Boc96] W. Bochtler: *Entwurf und Implementierung eines interaktiven Werkzeugs für den Entwurf verteilter Datenbanken*. Diplomarbeit, Universität Ulm, Fakultät für Informatik, 1996.

[BRD01a] T. Bauer, M. Reichert und P. Dadam: *Adaptives und verteiltes Workflow-Management.* In: *Proc. Datenbanksysteme in Büro, Technik und Wissenschaft,* Oldenburg, März 2001.

[BRD01b] T. Bauer, M. Reichert und P. Dadam: *Effiziente Übertragung von Prozessinstanzdaten in verteilten Workflow-Management-Systemen.* Informatik Forschung und Entwicklung, 16(2), Juni 2001.

[BS89] I.N. Bronstein und K.A. Semendjajew: *Taschenbuch der Mathematik.* Verlag Harri Deutsch und Thun, 1989.

[BS95] M. Böhm und W. Schulze: *Grundlagen von Workflow-Managementsystemen.* Wissenschaftliche Beiträge zur Informatik, Universität Dresden, 1995.

[Cas81] L.M. Casey: *Decentralised Scheduling.* The Australian Computer Journal, 13(2):58–63, Mai 1981.

[CCPP98] F. Casati, S. Ceri, B. Pernici und G. Pozzi: *Workflow Evolution.* Data & Knowledge Engineering, 24(3):211–238, 1998.

[CDK95] G. Coulouris, J. Dollimore und T. Kindberg: *Distributed Systems: Concepts and Design.* Addison-Wesley, 2. Auflage, 1995.

[CGP+96] F. Casati, P. Grefen, B. Pernici, G. Pozzi und G. Sánchez: *WIDE: Workflow Model and Architecture.* CTIT Technical Report 96-19, University of Twente, 1996.

[CGS97] S. Ceri, P. Grefen und G. Sánchez: *WIDE – A Distributed Architecture for Workflow Management.* In: *Proc. 7th Int. Workshop on Research Issues in Data Engineering,* Birmingham, April 1997.

[CK88] T.L. Casavant und J.G. Kuhl: *A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems.* IEEE Transactions on Software Engineering, SE-14(2):141–154, Februar 1988.

[CLR96] T.H. Cormen, C.E. Leiserson und R.L. Rivest: *Introduction to Algorithms.* MIT Press, 1996.

[Dad96] P. Dadam: *Verteilte Datenbanken und Client/Server-Systeme.* Springer-Verlag, 1996.

[DG94] W. Deiters und V. Gruhn: *The FUNSOFT Net Approach to Software Process Management.* Int. Journal in Software Engineering and Knowledge Engineering, 4(2):229–256, 1994.

[DGA+97] A. Dogac, E. Gokkoca, S. Arpinar, P. Koksal, I. Cingil, B. Arpinar, N. Tatbul, P. Karagoz, U. Halici und M. Altinel: *Design and Implementation of a Distributed Workflow Management System: METUFlow*. In: *Proc. NATO Advanced Study Institute on Workflow Management Systems and Interoperability,* S. 61–91, Istanbul, August 1997.

[DHL91] U. Dayal, M. Hsu und R. Ladin: *A Transactional Model for Long-Running Activities.* In: *Proc. 17th Int. Conf. on Very Large Data Bases,* S. 113–122, Barcelona, September 1991.

[DKM+97] S. Das, K. Kochut, J. Miller, A. Sheth und D. Worah: *ORBWork: A Reliable Distributed CORBA-based Workflow Enactment System for METEOR$_2$.* Technical Report #UGA-CS-TR-97-001, Department of Computer Science, University of Georgia, Februar 1997.

[DKR+95] P. Dadam, K. Kuhn, M. Reichert, T. Beuter und M. Nathe: *ADEPT: Ein integrierender Ansatz zur Entwicklung flexibler, zuverlässiger kooperierender Assistenzsysteme in klinischen Anwendungsumgebungen*. In: *Proc. GI/SI-Jahrestagung*, S. 677–686, Zürich, September 1995.

[DR98] P. Dadam und M. Reichert: *The ADEPT WfMS Project at the University of Ulm*. In: *Proc. 1st European Workshop on Workflow and Process Management - Workflow Management Research Projects*, Zürich, Oktober 1998.

[DRK00] P. Dadam, M. Reichert und K. Kuhn: *Clinical Workflows - The Killer Application for Process-oriented Information Systems?* In: *Proc. 4th Int. Conf. on Business Information Systems*, S. 36–59, Posen, April 2000.

[DS90] L. Davis und M. Steenstrup: *Genetic Algorithms and Simulated Annealing: An Overview*. In: *Genetic Algorithms and Simulated Annealing.* Morgan Kaufmann, 1990.

[DS97] M. Dierker und M. Sander: *Lotus Notes 4.5 und Domino - Integration von Groupware und Internet.* Addison-Wesley, 1. Auflage, 1997.

[EG96] J. Eder und H. Groiss: *Ein Workflow-Managementsystem auf der Basis aktiver Datenbanken*. In: J. Becker, G. Vossen (Herausgeber): *Geschäftsprozeßmodellierung und Workflow-Management.* Int. Thomson Publishing, 1996.

[EGL96a] J. Eder, H. Groiss und W. Liebhart: *Workflow Management and Databases.* In: *2ème Forum Int. d'Informatique Appliquée*, Tunis, März 1996.

[EGL96b] J. Eder, H. Groiss und W. Liebhart: *Workflow-Systeme im WWW*. In: *Proc. ADV Kongreß*, Wien, 1996.

[EGR91] C.A. Ellis, S.J. Gibbs und G.L. Rein: *Groupware - Some Issues and Experiences.* Communications of the ACM, 34(1):38–58, Januar 1991.

[EL96a] J. Eder und W. Liebhart: *Workflow Recovery*. In: *Proc. 1st IFCIS Int. Conf. on Cooperative Information Systems*, S. 124–134, Brussels, Juni 1996.

[EL96b] J. Eder und W. Liebhart: *Workflow Transactions.* In: *Workflow Handbook 1997. Handbook of the Workflow Management Coalition WfMC*. John Wiley, November 1996.

[ELZ86] D.L. Eager, E.D. Lazowska und J. Zahorjan: *Adaptive Load Sharing in Homogenous Distributed Systems.* IEEE Transactions on Software Engineering, SE-12(5):662–675, Mai 1986.

[Emd91] W. Emde: *Modellbildung, Wissensrevision und Wissensrepräsentation im Maschinellen Lernen.* Springer-Verlag, 1991.

[End98] H. Enderlin: *Realisierung einer verteilten Workflow-Ausführungskomponente auf Basis von IBM FlowMark.* Diplomarbeit, Universität Ulm, Fakultät für Informatik, 1998.

[GAC+97] E. Gokkoca, M. Altinel, I. Cingil, E.N. Tatbul, P. Koksal und A. Dogac: *Design and Implementation of a Distributed Workflow Enactment Service.* In: *Proc. 2nd IFCIS Conf. on Cooperative Information Systems*, S. 89–98, Kiawah Island, SC, Juni 1997.

[GB65] S. Golomb und L. Baumert: *Backtrack Programming.* Journal of the ACM, 12(4):516–524, 1965.

[GE96] H. Groiss und J. Eder: *Kooperation von Workflowsystemen im World-Wide Web.* In: *Proc. EMISA Jahrestagung,* S. 90–95, Aachen, 1996.

[GHS95] D. Georgakopoulos, M. Hornick und A. Sheth: *An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure.* Distributed and Parallel Databases, 3(2):119–152, April 1995.

[GJS+99] B. Gronemann, G. Joeris, S. Scheil, M. Steinfort und H. Wache: *Supporting Cross-Organizational Engineering Processes by Distributed Collaborative Workflow Management - The MOKASSIN Approach.* In: *Proc. 2nd Symposium on Concurrent Multidisciplinary Engineering, 3rd Int. Conf. on Global Engineering Networking,* Bremen, September 1999.

[GMPP97] A. Grasso, J.L. Meunier, D. Pagani und R. Pareschi: *Distributed Coordination and Workflow on the World Wide Web.* Computer Supported Cooperative Work: The Journal of Collaborative Computing, 6:175–200, 1997.

[GMWW99] M. Gillmann, P. Muth, G. Weikum und J. Weissenfels: *Benchmarking von Workflow-Management-Systemen.* In: *Proc. Datenbanksysteme in Büro, Technik und Wissenschaft,* S. 456–465, Freiburg, März 1999.

[Gos91] A. Goscinski: *Distributed Operating Systems: The Logical Design.* Addison-Wesley, 1991.

[GR93] J. Gray und A. Reuter: *Transaction Processing: Concepts and Techniques.* Morgan Kaufmann Publishers, 1993.

[Gri97] M. Grimm: *ADEPT$_{time}$: Temporale Aspekte in flexiblen Workflow-Management-Systemen.* Diplomarbeit, Universität Ulm, Fakultät für Informatik, 1997.

[Gro53] H.A. Grosch: *High Speed Arithmetic: The Digital Computer as a Research Tool.* Journal of the Optical Society of America, 43(4), April 1953.

[Gro75] H.A. Grosch: *Grosch's Law Revisited.* Computerworld, 8(16), April 1975.

[GTMG91] M. Gerla, T. Tai, J.A.S. Monteiro und G. Gallassi: *Interconnecting LANs and MANs to ATM.* In: *Proc. 16th Conf. on Local Computer Networks,* S. 259–270, 1991.

[GVBA97] P. Grefen, J. Vonk, E. Boertjes und P. Apers: *Two-Layer Transaction Management for Workflow Management Applications.* In: *Proc. 8th Int. Workshop on Database and Expert Systems Applications,* S. 430–439, Toulouse, September 1997.

[GWWK99] M. Gillmann, J. Weissenfels, G. Weikum und A. Kraiss: *Performance Assessment and Configuration of Enterprise-Wide Workflow Management Systems.* In: *Proc. Workshop Enterprise-wide and Cross-enterprise Workflow Management: Concepts, Systems, Applications, 29. Jahrestagung der GI,* S. 18–24, Paderborn, Oktober 1999.

[GWWK00] M. Gillmann, J. Weissenfels, G. Weikum und A. Kraiss: *Performance and Availability Assessment for the Configuration of Distributed Workflow Management Systems.* In: *Proc. 7th Int. Conf. on Extending Database Technology,* S. 183–201, Konstanz, März 2000.

[Ham95] M. Hammer: *The Reengineering Revolution - The Handbook.* Harper-Collins-Publishers, 1995.

[Hei00] C. Heinlein: *Workflow- und Prozeßsynchronisation mit Interaktionsausdrücken und -graphen – Konzeption und Realisierung eines Formalismus zur Spezifikation und Implementierung von Synchronisationsbedingungen*. Dissertation, Universität Ulm, Fakultät für Informatik, Juli 2000.

[Hei01] C. Heinlein: *Workflow and Process Synchronization with Interaction Expressions and Graphs.* In: *Proc. 17th Int. Conf. on Data Engineering,* April 2001.

[HIM96] M. Hesselmann, K. Irmscher und C. Mittasch: *Mobile Users in Workflow Management Applications.* In: O. Spaniol, J. Slavik, O. Drobnik (Herausgeber): *Personal Wireless Communications, Aachener Beiträge zur Informatik, Band 19,* S. 113–126, 1996.

[HK95] J. Herget und I. Kreitmeier: *Design von Geschäftsprozessen in semi-strukturierten Umgebungen: Das Eignungspotential von Groupware-Systemen für Workflow-Anwendungen.* In: *Proc. GI/SI-Jahrestagung,* S. 233–244, Zürich, September 1995.

[HM88] G. Held und T.R. Marshall: *Data Compression: Techniques and Applications, Hardware and Software Considerations.* John Wiley & Sons, 1988.

[Hof98] J. Hoffmann: *MATLAB und SIMULINK - Beispielorientierte Einführung in die Simulation dynamischer Systeme.* Addison-Wesley, 1998.

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

[HRB+00] C. Hensinger, M. Reichert, T. Bauer, T. Strzeletz und P. Dadam: *ADEPT$_{workflow}$ - Advanced Workflow Technology for the Efficient Support of Adaptive, Enterprise-wide Processes.* In: *7th Int. Conf. on Extending Database Technology, Software Demonstrations Track,* S. 29–30, Konstanz, März 2000.

[HS96] P. Heinl und H. Schuster: *Towards a Highly Scaleable Architecture for Workflow Management Systems.* In: *Proc. 7th Int. Workshop on Database and Expert Systems Applications,* S. 439–444, Zurich, September 1996.

[iABa] iABG: *Vorgangssteuerungssystem ProMInanD – Benutzerhandbuch, Version 1.7.*

[iABb] iABG: *Vorgangssteuerungssystem ProMInanD – Systemverwalterhandbuch, Version 1.8.*

[IBM94] IBM: *Encina Product Family Overview for AIX, Document Number: SC23-2443-04,* 5. Auflage, Oktober 1994.

[IBM96a] IBM: *FlowMark – Installation and Maintenance, Version 2 Release 2, Document Number: SH12-6260-00,* Februar 1996.

[IBM96b] IBM: *FlowMark – Modeling Workflow, Version 2 Release 2, Document Number: SH19-8241-01*, Februar 1996.

[Jab95] S. Jablonski: *Workflow-Management-Systeme: Modellierung und Architektur*. Thomson Publishing, 1995.

[Jab97] S. Jablonski: *Architektur von Workflow-Mangement-Systemen.* Informatik Forschung und Entwicklung, Themenheft Workflow-Management, 12(2):72–81, 1997.

[JBS97] S. Jablonski, M. Böhm und W. Schulze: *Workflow-Management: Entwicklung von Anwendungen und Systemen; Facetten einer neuen Technologie*. dpunkt-Verlag, 1997.

[JH92] J. Johnson und D. Hudson: *Die OLTP-Produkte für offene Systeme gewinnen an Marktreife*. Computerwoche, 12:49–54, März 1992.

[JH98] G. Joeris und O. Herzog: *Managing Evolving Workflow Specifications.* In: *Proc. 3rd IFCIS Conf. on Cooperative Information Systems*, New York, August 1998.

[JH99] G. Joeris und O. Herzog: *Towards Flexible and High-Level Modeling and Enacting of Processes.* In: *Proc. 11th Int. Conf. on Advanced Information Systems Engineering*, S. 88–102, Heidelberg, Juni 1999.

[Joe99] G. Joeris: *Defining Flexible Workflow Execution Behaviors*. In: *Proc. Workshop Enterprise-wide and Cross-enterprise Workflow Management: Concepts, Systems, Applications, 29. Jahrestagung der GI*, S. 49–55, Paderborn, Oktober 1999.

[KAD98] P. Koksal, S. Arpinar und A. Dogac: *Workflow History Management.* ACM SIGMOD-Record, 27(1):67–75, März 1998.

[KAGM96] M. Kamath, G. Alonso, R. Günthör und C. Mohan: *Providing High Availability in Very Large Workflow Management Systems.* In: *Proc. 5th Int. Conf. on Extending Database Technology*, S. 427–442, Avignon, März 1996.

[KAK+98] P. Karagoz, S. Arpinar, P. Koksal, N. Tatbul, E. Gokkoca und A. Dogac: *Task Handling in Workflow Management Systems.* In: *Proc. Int. Workshop on Issues and Applications of Database Technology*, S. 71–78, Berlin, Juni 1998.

[Kar94] B. Karbe: *Flexible Vorgangssteuerung mit ProMInanD.* In: U. Hasenkamp, S. Kirn, M. Syring (Herausgeber): *CSCW – Computer Supported Cooperative Work*, S. 117–133. Addison-Wesley, 1994.

[KK88] K.L. Kraemer und J.L. King: *Computer-Based Systems for Cooperative Work and Group Decision Making.* ACM Computing Surveys, 20(2):115–146, Juni 1988.

[Klö97] A. Klöpfer: *Vergleich verschiedener TP-Monitore unter konzeptionellen und anwendungsorientierten Aspekten.* Diplomarbeit, Universität Ulm, Fakultät für Informatik, 1997.

[KMO98] B. Kiepuszewski, R. Muhlberger und M.E. Orlowska: *FlowBack: Providing Backward Recovery for Workflow Management Systems*. In: *Proc. ACM SIGMOD Int. Conf. on Management of Data*, Seattle, Juni 1998.

[KR90a] B.H. Karbe und N.G. Ramsperger: *Influence of Execption Handling on the Support of Cooperative Office Work.* In: S. Gibbs, A.A. Verrijn-Stuart (Herausgeber): *Proc. IFIP WG8.4 Conf. on Multi-User Interfaces and Applications,* S. 355–370. Elsevier Science Publishers B.V., 1990.

[KR90b] B.W. Kernighan und D.M. Ritchie: *Programmieren in C.* Hanser, Prentice-Hall, 2. Auflage, 1990.

[KR91a] B. Karbe und N. Ramsperger: *Advanced Task Allocation in ProMInanD.* In: *Proc. 4th Int. Conf. on Human-Computer Interaction,* S. 1098–1102, Stuttgart, 1991.

[KR91b] B. Karbe und N. Ramsperger: *Concepts and Implementation of Migrating Office Processes.* In: W. Brauer, D. Hernandez (Herausgeber): *Verteilte Künstliche Intelligenz und kooperatives Arbeiten,* S. 136–147. Springer-Verlag, 1991.

[KR96] M. Kamath und K. Ramamritham: *Correctness Issues in Workflow Management.* Distributed Systems Engineering, 3(4):213–221, Dezember 1996.

[KRW90] B. Karbe, N. Ramsperger und P. Weiss: *Support of Cooperative Work by Electronic Circulation Folders.* In: *Proc. Conf. on Office Information Systems,* S. 109–117, Cambridge, MA, 1990.

[Kub98] M. Kubicek: *Organisatorische Aspekte in flexiblen Workflow-Management-Systemen.* Diplomarbeit, Universität Ulm, Fakultät für Informatik, 1998.

[LA94] F. Leymann und W. Altenhuber: *Managing Business Processes as an Information Resource.* IBM Systems Journal, 33(2):326–348, 1994.

[Lan92] H. Langendörfer: *Leistungsanalyse von Rechensystemen: Messen, Modellieren, Simulation.* Carl Hanser, 1992.

[Lee77] R.B.L. Lee: *Performance Bounds in Parallel Processor Organizations.* In: D.J. Kuck et. al. (Herausgeber): *High Speed Computing and Algorithm Organization.* Academic Press, 1977.

[Len97] R. Lenz: *Adaptive Datenreplikation in verteilten Systemen.* McGraw-Hill, 1997.

[Ley95] F. Leymann: *Supporting Business Transactions via Partial Backward Recovery in Workflow Managment Systems.* In: *Proc. Datenbanksysteme in Büro, Technik und Wissenschaft,* S. 51–70, Dresden, März 1995.

[Ley97] F. Leymann: *Transaktionsunterstützung für Workflows.* Informatik Forschung und Entwicklung, Themenheft Workflow-Management, 12(2):82–90, 1997.

[LH87] D.A. Lelewer und D.S. Hirschberg: *Data Compression.* ACM Computing Surveys, 19(3):261–296, September 1987.

[Lin95] D.S. Linthicum: *How to buy TP Monitors.* Open Computing, 12(7):66–67, Juli 1995.

[LK91] H. Lewe und H. Krcmar: *Groupware (Das aktuelle Schlagwort).* Informatik-Spektrum, 14:345–348, 1991.

[LR94] F. Leymann und D. Roller: *Business Process Management with FlowMark.* In: *Proc. IEEE Spring Computer Conf., San Francisco,* S. 230–234, März 1994.

[LR00] F. Leymann und D. Roller: *Production Workflow - Concepts and Techniques.* Prentice Hall, 2000.

[LW66] E.L. Lawler und D.W. Wood: *Branch-and-bound Methods: A Survey*. Operations Research, 14(4):699–719, 1966.

[MD94] C. Mohan und D. Dievendorff: *Recent Work on Distributed Commit Protokolls, and Recoverable Messaging and Queuing.* IEEE Computer Society, Bulletin of the Technical Committee on Data Engineering, 17(1):22–28, März 1994.

[Mit97] C. Mittasch: *BPAFrame Phase 2 – a Framework for Workflow Management.* In: *Proc. 14. ITG/GI-Fachtagung Architektur von Rechensystemen,* Rostock, September 1997.

[MIZ+96] C. Mittasch, K. Irmscher, T. Ziegert, T. Lodderstedt, S. Müller und K. Sommerfeld: *User Services in BPAFrame – a Framework for Workflow-Management-Systems*. In: N. Terashima, E. Altman (Herausgeber): *Advanced IT Tools*, S. 303–310. Chapman & Hall, 1996.

[ML77] H.L. Morgan und K.D. Levin: *Optimal Program and Data Locations in Computer Networks*. Comm. of the ACM, 20:315–322, 1977.

[MLO86] C. Mohan, B. Lindsay und R. Obermarck: *Transaction Management in the R^* Distributed Database Management System.* Transactions on Database Systems, 11(4):378–396, Dezember 1986.

[MMSL96] C. Mittasch, S. Müller, K. Sommerfeld und T. Lodderstedt: *Entwurf und Realisierung eines Frameworks für die Automatisierung von Geschäftsabläufen.* In: C. Mayr (Herausgeber): *Beherrschung von Informationssystemen, Proc. 26. Jahrestagung der GI,* S. 303–310. R. Oldenbourg, 1996.

[MP71] M. Minsky und S. Papert: *On some Associative Parallel and Analog Computations.* In: E.J. Jacks (Herausgeber): *Associative Information Techniques.* American Elsevier, 1971.

[MPS+98] J. Miller, D. Palaniswami, A. Sheth, K. Kochut und H. Singh: *WebWork: METEOR's Web-based Workflow Management System.* Journal of Intelligent Information Systems, Special Issue on Workflow Management Systems, 10(2):185–215, März/April 1998.

[MSKP97] J. Miller, A. Sheth, K. Kochut und D. Palaniswami: *The Future of Web-Based Workflows.* In: *Proc. Int. Workshop on Research Directions in Process Technology*, Nancy, Juli 1997.

[MSKW96] J.A. Miller, A.P. Sheth, K.J. Kochut und X. Wang: *CORBA-Based Run-Time Architectures for Workflow Management Systems.* Journal of Database Management, Special Issue on Multidatabases, 7(1):16–27, 1996.

[MWW+98] P. Muth, D. Wodtke, J. Weißenfels, A. Kotz-Dittrich und G. Weikum: *From Centralized Workflow Specification to Distributed Workflow Execution.* Journal of Intelligent

Information Systems, Special Issue on Workflow Management Systems, 10(2):159–184, März/April 1998.

[NCWD84] S. Navathe, S. Ceri, G. Wiederhold und J. Dou: *Vertical Partitioning Algorithms for Database Design.* ACM Transactions on Database Systems, 9(4):680–710, 1984.

[NR89] S.B. Navathe und M. Ra: *Vertical Partitioning for Database Design: A Graphical Algorithm.* In: *Proc. ACM SIGMOD Int. Conf. on Management of Data*, Band 18, S. 440–450, Portland, Juni 1989.

[NSS98] J. Neeb, R. Schamburger und H. Schuster: *Using Distributed Object Middleware to Implement Scalable Workflow Management Systems.* In: *Proc. Int. Workshop on Issues and Applications of Database Technology*, Berlin, Juli 1998.

[Obe96] A. Oberweis: *Modellierung und Ausführung von Workflows mit Petri-Netzen.* Teubner-Verlag, 1996.

[OMG95a] OMG: *CORBAservices: Common Object Services Specification.* Technischer Bericht, Object Management Group, März 1995.

[OMG95b] OMG: *Object Management Architecture Guide.* John Wiley & Sons, 3. Auflage, Juni 1995.

[OMG95c] OMG: *The Common Object Request Broker: Architecture and Specification.* Technischer Bericht Revision 2.0, Object Management Group, Juli 1995.

[Onv92] R.O. Onvural: *On Performance Characteristics of ATM Networks.* In: *Proc. IEEE Int. Conf. on Communications*, S. 1004–1008, Juni 1992.

[ÖV91] M.T. Özsu und P. Valduriez: *Principles of Distributed Database Systems.* Prentice Hall, 1991.

[OvG89] R.H.J.M. Otten und L.P.P.P. van Ginneken: *The Annealing Algorithm.* Kluwer Academic Publishers, 1989.

[Par94] C. Partridge: *Gigabyte Networking.* Addison-Wesley, 1994.

[Pic98] G. Piccinelli: *Distributed Workflow Management: The TEAM Model.* In: *Proc. 3rd IFCIS Int. Conf. on Cooperative Information Systems*, S. 292–299, New York, August 1998.

[Qui88] M.J. Quinn: *Algorithmenbau und Parallelcomputer.* McGraw-Hill, 1988.

[RBD99] M. Reichert, T. Bauer und P. Dadam: *Enterprise-Wide and Cross-Enterprise Workflow-Management: Challenges and Research Issues for Adaptive Workflows.* In: *Proc. Workshop Enterprise-wide and Cross-enterprise Workflow Management: Concepts, Systems, Applications, 29. Jahrestagung der GI*, S. 56–64, Paderborn, Oktober 1999.

[RBD00] M. Reichert, T. Bauer und P. Dadam: *ADEPT - Realisierung flexibler und zuverlässiger unternehmensweiter Workflow-Anwendungen.* In: *Proc. Conf. on Knowledge Engineering, Management, Consulting & Training*, Leipzig, September 2000.

[RD97] M. Reichert und P. Dadam: *A Framework for Dynamic Changes in Workflow Management Systems.* In: *Proc. 8th Int. Workshop on Database and Expert Systems Applications,* S. 42–48, Toulouse, September 1997.

[RD98] M. Reichert und P. Dadam: *ADEPT$_{flex}$ – Supporting Dynamic Changes of Workflows Without Losing Control.* Journal of Intelligent Information Systems, Special Issue on Workflow Management Systems, 10(2):93–129, März/April 1998.

[Rei00] M. Reichert: *Dynamische Ablaufänderungen in Workflow-Management-Systemen.* Dissertation, Universität Ulm, Fakultät für Informatik, Juli 2000.

[RHD98] M. Reichert, C. Hensinger und P. Dadam: *Supporting Adaptive Workflows in Advanced Application Environments.* In: *Proc. EDBT Workshop on Workflow Management Systems,* S. 100–109, Valencia, März 1998.

[RS94] M. Rusinkiewicz und A. Sheth: *Specification and Execution of Transactional Workflows.* In: W. Kim (Herausgeber): *Modern Database Systems: The Object Model, Interoperability, and Beyond,* S. 592–620. ACM Press, 1994.

[Rup95] W. Rupietta: *Flexible Geschäftsprozesse mit Workflow-Anwendungen.* In: *Geschäftsprozesse und Workflow-Systeme in der evolutionären Unternehmung.* Gesellschaft für Informatik, FA 5.2 Rundbrief: Informationssystem-Architekturen, Dezember 1995.

[RW94] W. Rupietta und G. Wernke: *Umsetzung organisatorischer Regelungen in der Vorgangsbearbeitung mit Workparty und ORM.* In: U. Hasenkamp, S. Kirn, M. Syring (Herausgeber): *CSCW – Computer Supported Cooperative Work,* S. 135–154. Addison-Wesley, 1994.

[SBCM95] G. Samaras, K. Britton, A. Citron und C. Mohan: *Two-Phase Commit Optimizations in a Commercial Distributed Environment.* Distributed and Parallel Databases, 3(4):325–360, Oktober 1995.

[Sch93] A. Schill: *DCE - Das OSF Distributed Environment, Einführung und Grundlagen.* Springer Verlag, 1993.

[Sch96] A.-W. Scheer: *ARIS-Toolset: Vom Forschungsprototypen zum Produkt.* Informatik-Spektrum, 19(2):71–78, April 1996.

[Sch97a] U. Schöning: *Algorithmen - kurz gefasst.* Spektrum Akademischer Verlag, 1997.

[Sch97b] H. Schuster: *Architektur verteilter Workflow-Management-Systeme.* Dissertation, Universität Erlangen-Nürnberg, 1997.

[Sch98] A.-W. Scheer: *ARIS - Modellierungsmethoden, Metamodelle, Anwendungen.* Springer-Verlag, 1998.

[She93] M. Sherman: *Architecture of the Encina Distributed Transaction Processing Family.* In: *Proc. ACM SIGMOD Int. Conf. on Management of Data,* S. 460–463, Mai 1993.

[Sie95a] Siemens Nixdorf: *WorkParty - Benutzerhandbuch, Version 2.0,* August 1995.

[Sie95b] Siemens Nixdorf: *WorkParty - der Geschäftsprozeßmanager. Geschäftsprozeßmanagement und integrierte Vorgangsbearbeitung beschleunigen den Informationsfluß*, Oktober 1995.

[Sie95c] Siemens Nixdorf: *WorkParty - Entwicklerhandbuch, Version 2.0*, August 1995.

[SK97] A. Sheth und K.J. Kochut: *Workflow Applications to Research Agenda: Scalable and Dynamic Work Coordination and Collaboration Systems.* In: *Proc. NATO Advanced Study Institute on Workflow Management Systems and Interoperability*, S. 12–21, Istanbul, August 1997.

[SM96] A. Schill und C. Mittasch: *Workflow Management Systems on Top of OSF DCE and OMG CORBA.* Distributed Systems Engineering, 3(4):250–262, Dezember 1996.

[SM97] A. Schill und C. Mittasch: *CodAlf: A Decentralized Workflow Management System on Top of OSF DCE and DC++.* In: *Proc. IEEE/IFIP 3rd Int. Symposium on Autonomous Decentralized Systems*, S. 205–212, Berlin, April 1997.

[SNS99] H. Schuster, J. Neeb und R. Schamburger: *A Configuration Management Approach for Large Workflow Management Systems.* In: *Proc. Int. Joint Conf. on Work Activities Coordination and Collaboration*, S. 177–186, San Francisco, Februar 1999.

[SR93] A. Sheth und M. Rusinkiewicz: *On Transactional Workflows.* IEEE Computer Society, Bulletin of the Technical Committee on Data Engineering, 16(2):37–40, Juni 1993.

[Sta84] J.A. Stankovic: *Simulations of Three Adaptive, Decentralized Controlled, Job Scheduling Algorithms.* Computer Networks, 8(3):199–217, Juni 1984.

[Sto88] J.A. Storer: *Data Compression: Methods and Theory*. Computer Science Press, 1988.

[SVBC91] G.I. Stassinopoulos, I.S. Venieris, A. Bricca und R. Carli: *Bridged LAN Interconnection Through ATM*. In: *Proc. 9th Annual European Fibre Optic Communications and Local Area Network Conf.*, S. 302–309, Juni 1991.

[Tan92] A.S. Tanenbaum: *Computer-Netzwerke*. Wolfram's Fachverlag, 1992.

[Tur96] V. Turau: *Algorithmische Graphentheorie.* Addison-Wesley, 1996.

[UNI92] UNIX System Laboratories: *TUXEDO System Release 4.1 – Product Overview and Master Index.* Prentice Hall, 1992.

[VB96] G. Vossen und J. Becker: *Geschäftsprozeßmodellierung und Workflow-Management - Modelle, Methoden, Werkzeuge*. Int. Thomson Publishing, 1996.

[VE92] P. Vogel und R. Erfle: *Backtracking Office Procedures*. In: A.M. Tjo, I. Ramos (Herausgeber): *Database and Expert Systems Applications*, S. 506–511. Springer-Verlag, 1992.

[VWW96a] G. Vossen, M. Weske und G. Wittowski: *Dynamic Workflow Management on the Web*. Schriften zur Angewandten Mathematik und Informatik 24/96-I, Universität Münster, November 1996.

[VWW96b] G. Vossen, M. Weske und G. Wittowski: *Prototypische Implementierung von WASA: Flexibles und plattformunabhängiges Workflow-Management.* Schriften zur Angewandten Mathematik und Informatik 23/96-I, Universität Münster, September 1996.

[Web98] M. Weber: *Verteilte Systeme.* Spektrum Akademischer Verlag, 1998.

[Wel83] P.D. Welch: *The Statistical Analysis of Simulation Results.* In: S.S. Lavenberg (Herausgeber): *Computer Performance Modeling Handbook*, S. 267–329. Academic Press, 1983.

[Wes98] M. Weske: *Flexible Modeling and Execution of Workflow Activities.* In: *Proc. 31st Hawaii Int. Conf. on System Sciences*, S. 713–722, Hawaii, 1998.

[Wes99a] M. Weske: *Adaptive Workflows based in Flexible Assignment of Workflow Schemas and Workflow Instances.* In: *Proc. Workshop Enterprise-wide and Cross-enterprise Workflow Management: Concepts, Systems, Applications, 29. Jahrestagung der GI*, S. 42–48, Paderborn, Oktober 1999.

[Wes99b] M. Weske: *State-based Modeling of Flexible Workflow Executions in Distributed Environments.* Journal of Integrated Design and Process Science, 1999.

[Wes99c] M. Weske: *Workflow Management Through Distributed and Persistent CORBA Workflow Objects.* In: *Proc. 11th Int. Conf. on Advanced Information Systems Engineering*, Heidelberg, 1999.

[WHKS98] M. Weske, J. Hündling, D. Kuropka und H. Schuschel: *Objektorientierter Entwurf eines flexiblen Workflow-Management-Systems.* Informatik Forschung und Entwicklung, 13(4):179–195, 1998.

[WK96] J. Wäsch und W. Klas: *History Merging as a Mechanism for Concurrency Control in Cooperative Environments.* In: *Proc. 6th Int. Workshop on Research Issues in Data Engineering – Interoperability of Nontraditional Database Systems*, S. 76–85, New Orleans, Februar 1996.

[WKM+95] D. Wodtke, A. Kotz-Dittrich, P. Muth, M. Sinnwell und G. Weikum: *Mentor: Entwurf einer Workflow-Management-Umgebung basierend auf State- und Aktivitycharts.* In: *Proc. Datenbanksysteme in Büro, Technik und Wissenschaft*, März 1995.

[WM85] Y.-T. Wang und R.J.T. Morris: *Load Sharing in Distributed Systems.* IEEE Transactions on Computers, C-34(3):204–217, März 1985.

[WMC99] Workflow Management Coalition: *Terminology & Glossary, Document Number WFMC-TC-1011, Document Status - Issue 3.0*, Februar 1999.

[WS97] D. Worah und A. Sheth: *Transactions in Transactional Workflows.* In: S. Jajodia, L. Kerschberg (Herausgeber): *Advanced Transaction Models and Architectures.* Kluwer Academic Publishers, 1997.

[WWK+97] G. Weikum, D. Wodtke, A. Kotz-Dittrich, P. Muth und J. Weißenfels: *Spezifikation, Verifikation und verteilte Ausführung von Workflows in MENTOR.* Informatik Forschung und Entwicklung, Themenheft Workflow-Management, 12(2):61–71, 1997.

[WWWK96a] J. Weißenfels, D. Wodtke, G. Weikum und A. Kotz-Dittrich: *The Mentor Architecture for Enterprise-wide Workflow Management.* In: *Proc. NSF Workshop on Workflow and Process Automation in Information Systems,* S. 69–73, Athens, Mai 1996.

[WWWK96b] D. Wodtke, J. Weißenfels, G. Weikum und A. Kotz-Dittrich: *The Mentor Project: Steps Towards Enterprise-Wide Workflow Management*. In: *Proc. 12th IEEE Int. Conf. on Data Engineering,* S. 556–565, New Orleans, LA, März 1996.

[Zei99] J. Zeitler: *Integration von Verteilungskonzepten in ein adaptives Workflow-Management-System.* Diplomarbeit, Universität Ulm, Fakultät für Informatik, 1999.

Teil IV

Anhang

Anhang A

Kurzschreibweisen

In diesem Teil des Anhangs werden die in der vorliegenden Arbeit verwendeten Kurzschreibweisen zusammengefasst. Er kann somit zum „Nachschlagen der Bedeutung“ von Kurzschreibweisen verwendet werden. So bietet die Tabelle A.1 eine Übersicht über in dieser Arbeit häufig verwendete Abkürzungen. In Tabelle A.2 sind die in dieser Arbeit – vor allem von den Algorithmen – verwendeten Funktionen aufgelistet (z.B. zur Bestimmung von Nachfolgerbeziehungen zwischen Aktivitäten). Schließlich wird in Tabelle A.3 die Bedeutung von mathematischen Ausdrücken, wie z.B. bestimmten Wahrscheinlichkeitsverteilungen, zusammengefasst. Damit die (evtl. ausführlichere) Originaldefinition eines Tabelleneintrags auf einfache Weise gefunden werden kann, ist der Abschnitt, in welchem die Kurzschreibweise eingeführt wurde, jeweils in spitzen Klammern angegeben. ⟨1.2.1⟩ bedeutet also, dass die entsprechende Kurzschreibweise im Abschnitt 1.2.1 definiert wurde.

API	Application Programming Interface ⟨7.2.4⟩
DB	Datenbank ⟨1.1.2.2⟩
DBMS	Datenbank-Management-System ⟨2.3.1⟩
FDL	FlowMark Definition Language ⟨11.1⟩
ID	Identifikation ⟨2.3.2.1⟩
LAN	Local Area Network ⟨1.2.2⟩
OE	Organisationseinheit ⟨1.1.2.1⟩
TRPC	Transactional Remote Procedure Call ⟨3.3.1⟩
WAN	Wide Area Network ⟨1.2.1⟩
WF	Workflow ⟨1.1.2⟩
WF-DB	Workflow-Datenbank ⟨1.1.2.3⟩
WfMS	Workflow-Management-System ⟨1⟩
WV	Wahrscheinlichkeitsverteilung ⟨4⟩
2PC	2-Phasen-Commit-Protokoll ⟨3.3.1⟩

Tabelle A.1: Übersicht über in dieser Arbeit verwendete Abkürzungen.

$\in$	$E \in H$ ist erfüllt, wenn der Historieneintrag E in der Ablaufhistorie H enthalten ist ⟨2.2.3.4⟩
$\cap$	$H \cap A$ sind die Einträge der Ablaufhistorie H, die zu in der Menge A enthaltenen Aktivitäteninstanzen gehören ⟨2.2.3.4⟩
$-$	$H - A$ sind die Einträge der Ablaufhistorie H, die zu in der Menge A nicht enthaltenen Aktivitäteninstanzen gehören ⟨2.2.3.4⟩
$ActivityInstance(E)$	zum Ablaufhistorieneintrag E gehörende Aktivitäteninstanz ⟨2.2.3.4⟩
$ActTyp(a)$	zur Aktivitäteninstanz a gehörender Aktivitätentyp ⟨2.2.3.1⟩
$Append(H, E)$	hängt den Eintrag E (hinten) an die Ablaufhistorie H an ⟨2.2.3.4⟩
$Endloop(n)$	zur Schleifenstartaktivität n gehörende Schleifenendeaktivität ⟨2.2.3.3⟩
$Join(n)$	zur Split-Aktivität n gehörende Join-Aktivität ⟨2.2.3.3⟩
$LastEntry(H)$	hinterster Eintrag der Ablaufhistorie H ⟨2.2.3.4⟩
$LastWriter(d, a)$	letzte Vorgängeraktivitäteninstanz der Aktivitäteninstanz a, welche das Datenelement d geschrieben hat ⟨2.3.2.4⟩
$Pred_{Typ}(n)$	Menge der direkten Vorgängeraktivitäten der Aktivität n bzgl. der in der Menge Typ spezifizierten Kantentypen ⟨2.2.3.2⟩
$Pred^{*}_{Typ}(n)$	Menge der direkten und indirekten Vorgängeraktivitäten der Aktivität n bzgl. der in der Menge Typ spezifizierten Kantentypen ⟨2.2.3.2⟩
$PredInst_{Typ}(a)$	Menge der direkten Vorgängeraktivitäteninstanzen der Aktivitäteninstanz a bzgl. der in der Menge Typ spezifizierten Kantentypen ⟨2.2.3.2⟩
$PredInst^{*}_{Typ}(a)$	Menge der direkten und indirekten Vorgängeraktivitäteninstanzen der Aktivitäteninstanz a bzgl. der in der Menge Typ spezifizierten Kantentypen ⟨2.2.3.2⟩
$Reader(d)$	Menge der Aktivitäteninstanzen, welche das Datenelement d gelesen haben ⟨2.3.2.4⟩
$ReadSet(a)$	Menge der von der Aktivitäteninstanz a gelesen Datenelemente ⟨2.3.2.4⟩
$Split(n)$	zur Join-Aktivität n gehörende Split-Aktivität ⟨2.2.3.3⟩
$Startloop(n)$	zur Schleifenendeaktivität n gehörende Schleifenstartaktivität ⟨2.2.3.3⟩
$Succ_{Typ}(n)$	Menge der direkten Nachfolgeraktivitäten der Aktivität n bzgl. der in der Menge Typ spezifizierten Kantentypen ⟨2.2.3.2⟩
$Succ^{*}_{Typ}(n)$	Menge der direkten und indirekten Nachfolgeraktivitäten der Aktivität n bzgl. der in der Menge Typ spezifizierten Kantentypen ⟨2.2.3.2⟩
$SuccInst_{Typ}(a)$	Menge der direkten Nachfolgeraktivitäteninstanzen der Aktivitäteninstanz a bzgl. der in der Menge Typ spezifizierten Kantentypen ⟨2.2.3.2⟩
$SuccInst^{*}_{Typ}(a)$	Menge der direkten und indirekten Nachfolgeraktivitäteninstanzen der Aktivitäteninstanz a bzgl. der in der Menge Typ spezifizierten Kantentypen ⟨2.2.3.2⟩

Tabelle A.2: Übersicht über in dieser Arbeit verwendete Funktionen mit Bezug zum ADEPT-Basismodell.

$AnteilAusf_n(oe)$	Anteil, den die Benutzer der OE oe an den Ausführungen der Aktivität n haben ⟨5.3.2.1⟩
$AnteilBearb_n(oe)$	Anteil, den die Benutzer der OE oe an der Gesamtheit der Bearbeiter der Aktivität n darstellen ⟨5.3.2.1⟩
$ActorProb_n(D\|s)$	Anteil der Bearbeiter des Domains D an der Gesamtheit der Bearbeiter von Aktivität n, wenn Aktivität n vom Server s kontrolliert wird ⟨5.3⟩
$BearbZuordn_n$	Bearbeiterzuordnung der Aktivität n ⟨2.4⟩
$Dep(n,m)$	Art der Anhängigkeit zwischen den Bearbeitern der Aktivitäten m und n (B: selber Bearbeiter, OE: selbe OE, $NULL$: keine Abhängigkeit) ⟨5.1.4.2⟩
$DepMigrProb_{m,n}(s_2\|s_1)$	Wahrscheinlichkeit für eine Migration zum Server s_2 (beim Übergang von der Aktivität m zu n), wenn m vom Server s_1 kontrolliert wird ⟨5.4.2⟩
$DepServProb_n(s\|u)$	Wahrscheinlichkeit, dass Aktivität n vom Server s kontrolliert wird, wobei der Bearbeiter u dieser Aktivität vorgegeben ist ⟨5.3.2.2⟩
$Domain(x)$	der Domain, dem der Benutzer bzw. Server x zugeordnet ist ⟨3.1.4⟩
$ExProb_n(s,D)$	Wahrscheinlichkeit, dass Aktivität n vom Server s kontrolliert und von einem Bearbeiter im Domain D ausgeführt wird ⟨4.3⟩
$Ext\_in_n(l)$, $Ext\_out_n(l)$	von der Aktivität n mit externen Datenquellen im Domain l ausgetauschte Ein-, Ausgabedatenmenge ⟨4.2.2.3⟩
$G_n(u)$	individuelles Gewicht des Bearbeiters u bei der Bearbeitung der Aktivität n ⟨4.2.2.2⟩
$G_n(oe)$	zusätzliches Gewicht für die Bearbeiter der OE oe bei der Ausführung der Aktivität n ⟨5.3.2.1⟩
$Gewicht_n(u)$	Gesamtgewicht, mit welchem der Bearbeiter u bei der Ausführung der Aktivität n gewichtet wird ⟨5.3.2.1⟩
$In\_parameter\_size_n$	Größe der Eingabeparameterdaten von Aktivität n ⟨4.2.2.3⟩
$Migration\_size_{m,n}$	Datenvolumen für eine Migration von der Aktivität m zur Aktivität n ⟨4.2.2.3⟩
$MigrProb_{m,n}(s_1,s_2)$	Wahrscheinlichkeit, dass beim Übergang von Aktivität m nach n vom WF-Server s_1 zum Server s_2 migriert werden muss ⟨4.3⟩
$Out\_parameter\_size_n$	Größe der Ausgabeparameterdaten von Aktivität n ⟨4.2.2.3⟩
$PotBearb_n$	Menge der potentiellen Bearbeiter der Aktivität n ⟨2.4.3.2⟩
$PotServZuordn_n$	Menge der für Aktivität n potentiell möglichen Serverzuordnungen ⟨4.5⟩
$RefAllowed(CFS,n,m)$	prüft, ob die Aktivität m in $ServZuordn_n$ referenziert werden darf ⟨5.1.4.1⟩
$ReferencedAct_n$	Aktivität, die in $ServZuordn_n$ referenziert wird ⟨5.1.2⟩
$RelevantAct_n$	Aktivitäten, die für eine Referenzierung in $ServZuordn_n$ geeignet und relevant sind ⟨5.1.4.2⟩
$Server_n$	Server, der die Aktivitäteninstanz n kontrolliert ⟨3.2.3⟩
$ServProb_n(s)$	Wahrscheinlichkeit, dass die Aktivität n vom Server s (in Teilnetz s) kontrolliert wird ⟨5.3⟩
$ServZuordn_n$	Serverzuordnung der Aktivität n ⟨3.2.3⟩
$\#User_n(D\|s)$	Anzahl der potentiellen Bearbeiter von Aktivität n im Domain D, wenn die Aktivität vom Server s kontrolliert wird ⟨4.3⟩
$\#User_n\|s$	Anzahl der Bearbeiter, die Aktivität n bearbeiten dürfen, falls sie vom Server s kontrolliert wird ⟨5.3⟩
$WL\_delete\_size_n$	Datenvolumen für das Löschen eines Arbeitslisteneintrags der Aktivität n ⟨4.2.2.3⟩
$WL\_insert\_size_n$	Datenvolumen für das Einfügen eines Arbeitslisteneintrags der Aktivität n ⟨4.2.2.3⟩

Tabelle A.3: Übersicht über in dieser Arbeit verwendete Ausdrücke mit Bezug zur Zuordnung von Bearbeitern und WF-Servern zu Aktivitäten.

Anhang B

Beweise

Im diesem Teil des Anhangs werden einige der in dieser Arbeit gemachten Aussagen bewiesen. Diese wurden aus dem Hauptteil der Arbeit ausgegliedert, um dessen „Lesefluss“ nicht zu verschlechtern.

B.1 Korrektheit von Algorithmus 5.9

Um die Korrektheit von Algorithmus 5.9 zeigen zu können, benötigen wir Lemma B.1. Dieses sagt aus, dass das Gesamtgewicht aller Bearbeiter einer Aktivität n durch die Multiplikation mit $Gewicht_n(u)$ in Algorithmus 5.10 nicht verändert wird. Es kommt lediglich zu Verschiebungen zwischen den verschiedenen OE, d.h. die Summe der Gewichte der Bearbeiter einer OE kann sich verändern.

Lemma B.1 $\sum_{u \in PotBearb_n} Gewicht_n(u) = \sum_{u \in PotBearb_n} G_n(u)$

Beweis:

$$\sum_{u \in PotBearb_n} Gewicht_n(u)$$

$$= \sum_{oe} \sum_{\substack{u \in PotBearb_n \\ \wedge\ OE(u)=oe}} Gewicht_n(u)$$

$$= \sum_{oe} \sum_{\substack{u \in PotBearb_n \\ \wedge\ OE(u)=oe}} G_n(oe) \cdot G_n(u)$$

$$= \sum_{oe} G_n(oe) \cdot \sum_{\substack{u \in PotBearb_n \\ \wedge\ OE(u)=oe}} G_n(u)$$

$$= \sum_{oe} \frac{AnteilAusf_n(oe)}{AnteilBearb_n(oe)} \cdot \sum_{\substack{u \in PotBearb_n \\ \wedge\ OE(u)=oe}} G_n(u)$$

$$= \sum_{oe} \frac{AnteilBearb_m(oe)}{AnteilBearb_n(oe)} \cdot \sum_{\substack{u \in PotBearb_n \\ \wedge\ OE(u)=oe}} G_n(u)$$

mit $BearbZuordn_n$ ist (indirekt) von Aktivität m abhängig ($Dep(n,m) \in \{B, OE\}$)
und $BearbZuordn_m$ ist eine unabhängige Bearbeiterzuordnung

d.h.: $\forall$ OE oe: $AnteilAusf_n(oe) = AnteilAusf_m(oe) = AnteilBearb_m(oe)$

$$= \sum_{oe} \frac{AnteilBearb_m(oe)}{\sum\limits_{\substack{u \in PotBearb_n \\ \wedge\ OE(u)=oe}} G_n(u) \quad / \sum\limits_{u \in PotBearb_n} G_n(u)} \cdot \sum_{\substack{u \in PotBearb_n \\ \wedge\ OE(u)=oe}} G_n(u)$$

$$= \sum_{oe} \frac{AnteilBearb_m(oe)}{1 \;/ \sum\limits_{u \in PotBearb_n} G_n(u)}$$

$$= \sum_{u \in PotBearb_n} G_n(u) \cdot \sum_{oe} AnteilBearb_m(oe)$$

$$= \sum_{u \in PotBearb_n} G_n(u) \cdot \sum_{oe} \Big(\sum_{\substack{u \in PotBearb_m \\ \wedge\ OE(u)=oe}} G_m(u) \quad / \sum_{u \in PotBearb_m} G_m(u) \Big)$$

$$= \sum_{u \in PotBearb_n} G_n(u) \cdot \frac{1}{\sum\limits_{u \in PotBearb_m} G_m(u)} \cdot \sum_{oe} \sum_{\substack{u \in PotBearb_m \\ \wedge\ OE(u)=oe}} G_m(u)$$

$$= \sum_{u \in PotBearb_n} G_n(u) \cdot \frac{1}{\sum\limits_{u \in PotBearb_m} G_m(u)} \cdot \sum_{u \in PotBearb_m} G_m(u)$$

$$= \sum_{u \in PotBearb_n} G_n(u)$$

□

Mit Hilfe von Lemma B.1 ist es nun möglich, Lemma B.2 zu beweisen. Lemma B.2 besagt, dass der Anteil der Bearbeiter der Organisationseinheit oe an der Aktivität n (jeweils mit $Gewicht_n(u)$ gewichtet) dem Anteil dieser OE an der Ausführung dieser Aktivität ($AnteilAusf_n(oe)$) entspricht. Da alle Bearbeiter einer OE mit demselben zusätzlichen Faktor $G_n(oe)$ gewichtet werden, kann es innerhalb dieser Bearbeiter zu keinen „Verzerrungen“ der Gewichte kommen. Damit ist gezeigt, dass die Berechnung von $Gewicht_n(u)$ in Algorithmus 5.9 auf korrekte Art und Weise erfolgt.

Lemma B.2 Auch wenn eine Vorgängeraktivität die OE der Bearbeiter von Aktivität n determiniert, so gilt $\forall$ OE oe:

$$\sum_{\substack{u \in PotBearb_n \\ \wedge\ OE(u)=oe}} Gewicht_n(u) \quad / \sum_{u \in PotBearb_n} Gewicht_n(u) \;=\; AnteilAusf_n(oe)$$

Beweis: $\forall$ OE oe gilt:

$$\sum_{\substack{u \in PotBearb_n \\ \wedge\ OE(u)=oe}} Gewicht_n(u) \quad / \sum_{u \in PotBearb_n} Gewicht_n(u)$$

$$\overset{Lemma\ B.1}{=} \sum_{\substack{u \in PotBearb_n \\ \wedge\ OE(u)=oe}} Gewicht_n(u) \quad / \sum_{u \in PotBearb_n} G_n(u)$$

$$= \sum_{\substack{u \in PotBearb_n \\ \wedge\ OE(u)=oe}} G_n(oe) \cdot G_n(u) \quad / \sum_{u \in PotBearb_n} G_n(u)$$

$$= \sum_{\substack{u \in PotBearb_n \\ \wedge\ OE(u)=oe}} \frac{AnteilAusf_n(oe)}{AnteilBearb_n(oe)} \cdot G_n(u) \quad / \sum_{u \in PotBearb_n} G_n(u)$$

$$= \frac{AnteilAusf_n(oe)}{AnteilBearb_n(oe)} \cdot \sum_{\substack{u \in PotBearb_n \\ \wedge\ OE(u)=oe}} G_n(u) \quad / \sum_{u \in PotBearb_n} G_n(u)$$

$$= \frac{AnteilAusf_n(oe)}{\sum\limits_{\substack{u \in PotBearb_n \\ \wedge\ OE(u)=oe}} G_n(u) \;/\; \sum\limits_{u \in PotBearb_n} G_n(u)} \cdot \sum_{\substack{u \in PotBearb_n \\ \wedge\ OE(u)=oe}} G_n(u) \quad / \sum_{u \in PotBearb_n} G_n(u)$$

$$= AnteilAusf_n(oe)$$ □

B.2 Korrektheit von Algorithmus 5.17

Lemma B.3 zeigt, dass in den For-Schleifen von Algorithmus 5.17 alle relevanten Fälle berücksichtigt werden. Der Grund dafür ist, dass die Summe der Wahrscheinlichkeiten (für alle betrachteten Kombinationen von Bearbeitern der Aktivitäten m und n) den Wert 1 ergibt. Dieser Wert ist korrekt, weil er der Wahrscheinlichkeit entspricht, dass die Aktivitäten m und n von genau einem Benutzerpaar bearbeitet werden.

Lemma B.3 $\sum\limits_{u_1 \in PotBearb_m} \sum\limits_{u_2 \in Bearb_{u_1}} \left(\frac{Gewicht_m(u_1)}{Gesamtgewicht_1} \cdot \frac{Gewicht_n(u_2)}{Gesamtgewicht_2(u_1)} \right) = 1$

Beweis:

$$\sum_{u_1 \in PotBearb_m} \sum_{u_2 \in Bearb_{u_1}} \left(\frac{Gewicht_m(u_1)}{Gesamtgewicht_1} \cdot \frac{Gewicht_n(u_2)}{Gesamtgewicht_2(u_1)} \right)$$

$$= \sum_{u_1 \in PotBearb_m} \left(\frac{Gewicht_m(u_1)}{Gesamtgewicht_1} \cdot \sum_{u_2 \in Bearb_{u_1}} \frac{Gewicht_n(u_2)}{Gesamtgewicht_2(u_1)} \right)$$

$$= \frac{1}{Gesamtgewicht_1} \cdot \underbrace{\sum_{u_1 \in PotBearb_m} \left(Gewicht_m(u_1) \cdot \frac{1}{Gesamtgewicht_2(u_1)} \cdot \underbrace{\sum_{u_2 \in Bearb_{u_1}} Gewicht_n(u_2)}_{Gesamtgewicht_2(u_1)} \right)}_{Gesamtgewicht_1}$$

$$= 1$$ □

Außerdem gilt analog $\sum_{u_2 \in Bearb_{u_1}} \frac{Gewicht_n(u_2)}{Gesamtgewicht_2(u_1)} = 1$, so dass die Gewichtung des Bearbeiters u_1 von Aktivität m nicht durch die Multiplikationen mit $\frac{Gewicht_n(u_2)}{Gesamtgewicht_2(u_1)}$ verändert wird.

Das Gesamtgewicht eines Bearbeiterpaares u_1 und u_2 wird durch die Multiplikation mit dem Faktor $DepServProb_m(s_1|u_1) \cdot DepServProb_n(s_2|u_2)$ nicht beeinflusst, weil wegen $\sum_{s_1} DepServProb_m(s_1|u_1) = \sum_{s_2} DepServProb_n(s_2|u_2) = 1$ gilt:
$\sum_{s_1} \sum_{s_2} DepServProb_m(s_1|u_1) \cdot DepServProb_n(s_2|u_2) = 1$.

Anhang C

Simulation „Kreditantragsbearbeitung“

Im Folgenden wird der in Abschnitt 1.2.2 und 9.3 verwendete Kreditantragsbearbeitungs-WF detailliert beschrieben. Dazu werden im Abschnitt C.1 alle Eingangsparameter des Szenarios aufgelistet. In Abschnitt C.2 werden dann die Simulationsergebnisse vorgestellt und erläutert.

C.1 Eingangsparameter

In diesem Abschnitt wird das Szenario beschrieben, das bei der Simulation des in Abb. 1.3 und 9.8 dargestellten Kreditantrags-WF verwendet wurde. Dabei werden alle Simulationsparameter angegeben, unabhängig davon, ob sie einen Einfluss auf das Simulationsergebnis haben oder nicht.

C.1.1 Allgemeine Simulationsparameter

- Simulationsdauer:
  160000 sec
- Dauer Einschwingphase:
  120000 sec
- Anzahl Teilnetze:
  31 (Teilnetz 0: Bankzentrale, Teilnetz 1 . . . 30: Filialen 1 . . . 30)
- Datenvolumen für die Übertragung einer Arbeitsliste mit n Einträgen:
  20 Bytes + $n \cdot 40$ Bytes
- Verzögerung, bis ein Benutzer nach Beendigung einer Aktivität den nächsten Eintrag aus seiner Arbeitsliste auswählt[1]:
  1 sec
- Verzögerung, bis ein Benutzer, nach erfolglosem Versuch einen Eintrag aus seiner Arbeitsliste auszuwählen (bei leerer Arbeitsliste), dies ein zweites Mal versucht[1]:
  5 sec

[1] Die Verzögerungen wurden bei dieser Simulation bewusst extrem klein gewählt, damit eine möglichst hohe Auslastung der Bearbeiter erreicht werden kann. Nur dadurch konnte eine Netzlast erreicht werden, die sehr nahe am „rechnerischen Wert“ von 16 Mbit/sec (vgl. Abschnitt 1.2.2) liegt, da bei der Berechnung überhaupt keine Verzögerungen berücksichtigt werden.

C.1.2 Benutzer des WfMS

- Anzahl Benutzer:
  320
- Minimaler Zeitabstand[1] zwischen zwei Aktualisierungen der Arbeitsliste eines Benutzers (falls eine Veränderung stattgefunden hat, vgl. Abschnitt 2.5.3):
  10 sec

Die 320 Benutzer verteilen sich wie folgt auf die Bankzentrale und die Filialen.

- Zentrale:
  20 Benutzer in Teilnetz 0 mit Rolle AngestZentrale und OE Zentrale
- Zweigstelle $i = 1 \ldots 30$:
  je 10 Benutzer in Teilnetz i mit Rolle AngestFiliale und OE Filiale_i

C.1.3 Simulierter Workflow

- Der simulierte WF besteht aus einer Sequenz der folgenden Aktivitäten (vgl. Abb. 1.3 und Abb. 9.8):
  1. Dokumente einscannen:

Bearbeiter:	Starter der WF-Instanz
Größe Eingabedaten:	0 kB
Größe Ausgabedaten:	2 MB
Dauer:	300 sec $\pm$ 60 sec (gleichverteilt im Intervall)
Zeitabstand zur Nachfolgeraktivität:	0 sec
Datenvolumen für Migration:	2 MB

  2. Kreditantrag speichern:

Bearbeiter:	Rolle AngestZentrale
Größe Eingabedaten:	19 kB
Größe Ausgabedaten:	1 kB
Dauer:	30 sec $\pm$ 10 sec
Zeitabstand zur Nachfolgeraktivität:	0 sec
Datenvolumen für Migration:	2 MB

  3. Dokumente überprüfen:

Bearbeiter:	Rolle AngestFiliale, OE wie Aktivität 1
Größe Eingabedaten:	2 MB
Größe Ausgabedaten:	10 kB
Dauer:	300 sec $\pm$ 60 sec
Zeitabstand zur Nachfolgeraktivität:	0 sec
Datenvolumen für Migration:	2 MB

4. Kreditrisiko abschätzen:
   Bearbeiter: Rolle AngestFiliale, OE wie Aktivität 1
   Größe Eingabedaten: 2 MB
   Größe Ausgabedaten: 10 kB
   Dauer: 300 sec ± 60 sec
   Zeitabstand zur Nachfolgeraktivität: 0 sec
   Datenvolumen für Migration: 2 MB

5. (Kreditantrag) annehmen oder ablehnen:
   Bearbeiter: Rolle AngestFiliale, OE wie Aktivität 1
   Größe Eingabedaten: 2 MB
   Größe Ausgabedaten: 10 kB
   Dauer: 300 sec ± 60 sec
   Zeitabstand zur Nachfolgeraktivität: 0 sec
   Datenvolumen für Migration: 2 MB

6. Entscheidung speichern:
   Bearbeiter: Rolle AngestZentrale
   Größe Eingabedaten: 19 kB
   Größe Ausgabedaten: 1 kB
   Dauer: 30 sec ± 10 sec
   Zeitabstand zur Nachfolgeraktivität: 0 sec
   Datenvolumen für Migration: 2 MB

- Den Aktivitäten wurden wie folgt die WF-Server aus den Teilnetzen 0 . . . 30 zugeordnet.
  Zentrale WF-Steuerung: alle Aktivitäten werden vom WF-Server im Teilnetz 0 (Bankzentrale) kontrolliert.
  Verteilte WF-Steuerung: alle Aktivitäten werden vom dem WF-Server gesteuert, bei dem die WF-Instanz gestartet wurde (involvierte Bankfiliale).

- Anzahl WF-Instanzen:
  39000
  Da die 300 Bearbeiter der Bankfilialen für die Bearbeitung einer Aktivität durchschnittlich 300 sec benötigen und ein WF 4 für diese Bearbeiter relevante Aktivitäten enthält, können sie in der Simulationsdauer von 160000 sec eigentlich $\frac{160000 \cdot 300}{300 \cdot 4}$ = 40000 WF-Instanzen bearbeiten. Da bei einer vollständigen Auslastung der Bearbeiter aber deren Arbeitslisten überlaufen würden, wurde eine um 2,5% geringere Last simuliert. Dadurch ergeben sich 39000 WF-Instanzen. Die 20 Bearbeiter der Zentrale können während der Simulationsdauer $\frac{160000 \cdot 20}{30 \cdot 2}$ = 53300 WF-Instanzen bearbeiten, so dass diese Bearbeiter keinen Engpass darstellen.

C.2 Ergebnis der Simulation

In den nun folgenden Abschnitten wird das Simulationsergebnis vorgestellt und erläutert. Um dieses zu ermitteln, wurde das in Abschnitt C.1 beschriebene Szenario für den zentralen und für den verteilten Fall jeweils 10000 Mal simuliert.

C.2.1 Workflow-Management-System mit zentralem Server

In Tabelle C.1 ist die Belastung der WF-Server im zentralen Fall dargestellt. Dabei sind die folgenden Werte angegeben: die Gesamtbelastung aller WF-Server, die Belastung des Servers der Bankzentrale und die Last der Server der Filialen. Da diese Server im zentralen Fall nicht in die Kontrolle der WF-Instanzen involviert sind, ist ihre Last gleich Null und die Gesamtlast ist mit der des WF-Servers der Zentrale identisch.

Die Werte in den Tabellen C.1 bis C.4 geben das von der jeweiligen Komponente im eingeschwungenen Zustand durchschnittlich umgesetzte Datenvolumen (Spalte $Komm.$) bzw. die Anzahl der von ihr ausgeführten Aktionen (andere Spalten) an. Der 1. Wert m eines Eintrags $\begin{smallmatrix} m \\ \pm x \\ \pm y\% \end{smallmatrix}$ spezifiziert den Mittelwert der jeweiligen Last der 10000 Simulationsläufe. Dieser Wert ist in kB/sec bzw. Aktionen/sec angegeben. Der 2. Wert x gibt das 90% Konfidenzintervall für den Erwartungswert an, das sich als $[m-x, m+x]$ ergibt. Wie groß dieses Konfidenzintervall im Vergleich zum Mittelwert ist, wird durch den y Wert angegeben, der sich als $y = x/m \cdot 100\%$ ergibt.

In Tabelle C.1 ist die Belastung der WF-Server im zentralen Fall dargestellt. Damit haben die Spalten die folgenden Bedeutungen:

Komm.: von dem jeweiligen WF-Server umgesetztes Datenvolumen in kB/sec

WF-Start: Anzahl der pro Sekunde zu startenden WF-Instanzen

WF-Ende: Anzahl der WF-Instanzen, die pro Sekunde beendet werden

Akt-Start: Anzahl der pro Sekunde zu startenden Aktivitäteninstanzen

Akt-Ende: Anzahl der Aktivitäteninstanzen, die pro Sekunde beendet werden

AL-Upd.: Anzahl der Arbeitslisten, die pro Sekunde aktualisiert werden müssen

Migrat.: Anzahl der pro Sekunde auszuführenden Migrationen (eingehende und ausgehende)

Aktionen: Anzahl der Aktionen, die pro Sekunde ausgeführt werden müssen, wenn die Aktionen *WF-Start* bis $Migrat.$ gemeinsam betrachtet werden

	Komm.	**WF-Start**	**WF-Ende**	**Akt-Start**	**Akt-Ende**	**AL-Upd.**	**Migrat.**	**Aktionen**
Summe	1958.77	0.2437	0.2425	1.4551	1.4551	11.2100	0	14.6065
Server	±0.138	±0.00004	±0.00002	±0.00010	±0.00010	±0.00113	±0	±0.00135
	±0.0070%	±0.0144%	±0.0072%	±0.0070%	±0.0070%	±0.0101%	–	±0.0092%
Server	1958.77	0.2437	0.2425	1.4551	1.4551	11.2100	0	14.6065
Zentrale	±0.138	±0.00004	±0.00002	±0.00010	±0.00010	±0.00113	±0	±0.00135
	±0.0070%	±0.0144%	±0.0072%	±0.0070%	±0.0070%	±0.0101%	–	±0.0092%
Server	0	0	0	0	0	0	0	0
Filiale	±0	±0	±0	±0	±0	±0	±0	±0
1 . . . 30	–	–	–	–	–	–	–	–

Tabelle C.1: Belastung der WF-Server im zentralen Fall.

In Tabelle C.2 ist die Belastung des Kommunikationssystems dargestellt, also die Belastung der Teilnetze (einzeln und gesamt) und der Gateways (nur gesamt). Bei dieser Betrachtung sind die Spalten $Komm.$ und $Aktionen$ von besonderer Bedeutung, da sie die von der jeweiligen Komponente umgesetzte Datenmenge bzw. die Anzahl der übertragenen Nachrichten beschreiben. Die Spalten *WF-Start* bis $Migrat.$ beschreiben die Anzahl der entsprechenden Aktionen, in welche die Komponente pro Sekunde involviert ist, also z.B. die Anzahl der Aktivitätenprogrammstarts, die über die Gateways laufen.

	Komm.	WF-Start	WF-Ende	Akt-Start	Akt-Ende	AL-Upd.	Migrat.	Aktionen
Summe	3907.59	0.2437	0.2425	2.4252	2.4252	20.4671	0	25.8038
Teil-	±0.275	±0.00004	±0.00002	±0.00017	±0.00017	±0.00223	±0	±0.00258
netze	±0.0070%	±0.0144%	±0.0072%	±0.0070%	±0.0070%	±0.0109%	–	±0.0100%
Summe	1948.82	0	0	0.9701	0.9701	9.2571	0	11.1973
Gate-	±0.137	±0	±0	±0.00007	±0.00007	±0.00111	±0	±0.00123
ways	±0.0070%	–	–	±0.0070%	±0.0070%	±0.0119%	–	±0.0110%
Teilnetz	1958.77	0.2437	0.2425	1.4551	1.4551	11.2100	0	14.6065
Zentrale	±0.138	±0.00004	±0.00002	±0.00010	±0.00010	±0.00113	±0	±0.00135
	±0.0070%	±0.0144%	±0.0072%	±0.0070%	±0.0070%	±0.0101%	–	±0.0092%
Teilnetz	64.97	0	0	0.0323	0.0323	0.3087	0	0.3734
Filiale	±0.033	±0	±0	±0.00002	±0.00002	±0.00026	±0	±0.00029
1	±0.0514%	–	–	±0.0510%	±0.0510%	±0.0840%	–	±0.0779%
Teilnetz	64.93	0	0	0.0323	0.0323	0.3084	0	0.3730
Filiale	±0.034	±0	±0	±0.00002	±0.00002	±0.00026	±0	±0.00029
2	±0.0519%	–	–	±0.0515%	±0.0515%	±0.0852%	–	±0.0789%
Teilnetz	64.96	0	0	0.0323	0.0323	0.3086	0	0.3733
Filiale	±0.033	±0	±0	±0.00002	±0.00002	±0.00026	±0	±0.00029
3	±0.0510%	–	–	±0.0506%	±0.0506%	±0.0834%	–	±0.0773%
Teilnetz	64.98	0	0	0.0323	0.0323	0.3087	0	0.3734
Filiale	±0.033	±0	±0	±0.00002	±0.00002	±0.00026	±0	±0.00029
4	±0.0507%	–	–	±0.0503%	±0.0503%	±0.0829%	–	±0.0768%
Teilnetz	65.00	0	0	0.0324	0.0324	0.3089	0	0.3736
Filiale	±0.033	±0	±0	±0.00002	±0.00002	±0.00026	±0	±0.00029
5	±0.0500%	–	–	±0.0496%	±0.0496%	±0.0826%	–	±0.0765%
Teilnetz	64.95	0	0	0.0323	0.0323	0.3085	0	0.3732
Filiale	±0.034	±0	±0	±0.00002	±0.00002	±0.00026	±0	±0.00029
6	±0.0521%	–	–	±0.0516%	±0.0517%	±0.0852%	–	±0.0790%
Teilnetz	64.96	0	0	0.0323	0.0323	0.3086	0	0.3733
Filiale	±0.033	±0	±0	±0.00002	±0.00002	±0.00026	±0	±0.00029
7	±0.0511%	–	–	±0.0506%	±0.0507%	±0.0832%	–	±0.0771%
Teilnetz	64.98	0	0	0.0323	0.0323	0.3088	0	0.3734
Filiale	±0.033	±0	±0	±0.00002	±0.00002	±0.00026	±0	±0.00029
8	±0.0508%	–	–	±0.0504%	±0.0504%	±0.0835%	–	±0.0774%
Teilnetz	64.98	0	0	0.0323	0.0323	0.3086	0	0.3733
Filiale	±0.033	±0	±0	±0.00002	±0.00002	±0.00026	±0	±0.00029
9	±0.0512%	–	–	±0.0507%	±0.0507%	±0.0841%	–	±0.0779%
Teilnetz	64.95	0	0	0.0323	0.0323	0.3086	0	0.3732
Filiale	±0.033	±0	±0	±0.00002	±0.00002	±0.00026	±0	±0.00029
10	±0.0515%	–	–	±0.0511%	±0.0511%	±0.0840%	–	±0.0779%
Teilnetz	64.94	0	0	0.0323	0.0323	0.3084	0	0.3731
Filiale	±0.034	±0	±0	±0.00002	±0.00002	±0.00026	±0	±0.00029
11	±0.0516%	–	–	±0.0512%	±0.0512%	±0.0846%	–	±0.0784%
Teilnetz	64.92	0	0	0.0323	0.0323	0.3083	0	0.3729
Filiale	±0.034	±0	±0	±0.00002	±0.00002	±0.00026	±0	±0.00030
12	±0.0522%	–	–	±0.0517%	±0.0517%	±0.0856%	–	±0.0793%

Tabelle C.2: Belastung des Kommunikationssystems im zentralen Fall.

	Komm.	WF-Start	WF-Ende	Akt-Start	Akt-Ende	AL-Upd.	Migrat.	Aktionen
Teilnetz	64.97	0	0	0.0323	0.0323	0.3086	0	0.3733
Filiale	±0.033	±0	±0	±0.00002	±0.00002	±0.00025	±0	±0.00029
13	±0.0504%	–	–	±0.0501%	±0.0500%	±0.0825%	–	±0.0765%
Teilnetz	64.95	0	0	0.0323	0.0323	0.3086	0	0.3732
Filiale	±0.034	±0	±0	±0.00002	±0.00002	±0.00026	±0	±0.00029
14	±0.0516%	–	–	±0.0512%	±0.0512%	±0.0845%	–	±0.0784%
Teilnetz	64.96	0	0	0.0323	0.0323	0.3086	0	0.3732
Filiale	±0.034	±0	±0	±0.00002	±0.00002	±0.00026	±0	±0.00029
15	±0.0516%	–	–	±0.0512%	±0.0513%	±0.0840%	–	±0.0780%
Teilnetz	64.96	0	0	0.0323	0.0323	0.3086	0	0.3733
Filiale	±0.033	±0	±0	±0.00002	±0.00002	±0.00026	±0	±0.00029
16	±0.0511%	–	–	±0.0506%	±0.0506%	±0.0841%	–	±0.0779%
Teilnetz	64.97	0	0	0.0323	0.0323	0.3087	0	0.3734
Filiale	±0.033	±0	±0	±0.00002	±0.00002	±0.00026	±0	±0.00029
17	±0.0509%	–	–	±0.0505%	±0.0504%	±0.0832%	–	±0.0772%
Teilnetz	64.96	0	0	0.0323	0.0323	0.3085	0	0.3732
Filiale	±0.033	±0	±0	±0.00002	±0.00002	±0.00026	±0	±0.00029
18	±0.0516%	–	–	±0.0512%	±0.0512%	±0.0847%	–	±0.0785%
Teilnetz	64.94	0	0	0.0323	0.0323	0.3083	0	0.3730
Filiale	±0.034	±0	±0	±0.00002	±0.00002	±0.00026	±0	±0.00029
19	±0.0516%	–	–	±0.0512%	±0.0512%	±0.0847%	–	±0.0785%
Teilnetz	64.98	0	0	0.0323	0.0323	0.3086	0	0.3733
Filiale	±0.033	±0	±0	±0.00002	±0.00002	±0.00026	±0	±0.00029
20	±0.0501%	–	–	±0.0497%	±0.0497%	±0.0828%	–	±0.0766%
Teilnetz	64.96	0	0	0.0323	0.0323	0.3086	0	0.3733
Filiale	±0.033	±0	±0	±0.00002	±0.00002	±0.00026	±0	±0.00029
21	±0.0509%	–	–	±0.0505%	±0.0505%	±0.0834%	–	±0.0773%
Teilnetz	64.96	0	0	0.0323	0.0323	0.3085	0	0.3732
Filiale	±0.033	±0	±0	±0.00002	±0.00002	±0.00026	±0	±0.00029
22	±0.0511%	–	–	±0.0506%	±0.0506%	±0.0838%	–	±0.0776%
Teilnetz	64.93	0	0	0.0323	0.0323	0.3083	0	0.3730
Filiale	±0.034	±0	±0	±0.00002	±0.00002	±0.00026	±0	±0.00029
23	±0.0518%	–	–	±0.0514%	±0.0514%	±0.0847%	–	±0.0786%
Teilnetz	64.97	0	0	0.0323	0.0323	0.3086	0	0.3733
Filiale	±0.033	±0	±0	±0.00002	±0.00002	±0.00026	±0	±0.00029
24	±0.0511%	–	–	±0.0507%	±0.0507%	±0.0837%	–	±0.0776%
Teilnetz	64.99	0	0	0.0324	0.0323	0.3088	0	0.3735
Filiale	±0.033	±0	±0	±0.00002	±0.00002	±0.00025	±0	±0.00029
25	±0.0501%	–	–	±0.0497%	±0.0497%	±0.0825%	–	±0.0764%
Teilnetz	64.95	0	0	0.0323	0.0323	0.3085	0	0.3732
Filiale	±0.034	±0	±0	±0.00002	±0.00002	±0.00026	±0	±0.00030
26	±0.0524%	–	–	±0.0520%	±0.0520%	±0.0858%	–	±0.0796%
Teilnetz	64.97	0	0	0.0323	0.0323	0.3085	0	0.3732
Filiale	±0.033	±0	±0	±0.00002	±0.00002	±0.00026	±0	±0.00029
27	±0.0514%	–	–	±0.0510%	±0.0510%	±0.0845%	–	±0.0783%

Tabelle C.2 (Fortsetzung)

	Komm.	WF-Start	WF-Ende	Akt-Start	Akt-Ende	AL-Upd.	Migrat.	Aktionen
Teilnetz	64.95	0	0	0.0323	0.0323	0.3085	0	0.3731
Filiale	±0.033	±0	±0	±0.00002	±0.00002	±0.00026	±0	±0.00029
28	±0.0512%	–	–	±0.0508%	±0.0508%	±0.0846%	–	±0.0784%
Teilnetz	64.97	0	0	0.0323	0.0323	0.3086	0	0.3733
Filiale	±0.033	±0	±0	±0.00002	±0.00002	±0.00026	±0	±0.00029
29	±0.0510%	–	–	±0.0505%	±0.0505%	±0.0837%	–	±0.0776%
Teilnetz	64.95	0	0	0.0323	0.0323	0.3085	0	0.3732
Filiale	±0.033	±0	±0	±0.00002	±0.00002	±0.00026	±0	±0.00029
30	±0.0514%	–	–	±0.0510%	±0.0510%	±0.0839%	–	±0.0778%

Tabelle C.2 (Fortsetzung)

C.2.2 Verteilte Workflow-Ausführung

In Tabelle C.3 bzw. C.4 ist die Belastung der WF-Server bzw. des Kommunikationssystems dargestellt, die auftritt, wenn der WF-Typ verteilt gesteuert wird. Da in diesem Fall auch die WF-Server der Bankfilialen involviert sind, ist hier (im Gegensatz zum Abschnitt C.2.1) auch deren Belastung angegeben.

	Komm.	WF-Start	WF-Ende	Akt-Start	Akt-Ende	AL-Upd.	Migrat.	Aktionen
Summe	1958.91	0.2437	0.2425	1.4552	1.4551	18.1122	0	21.5087
Server	±0.140	±0.00004	±0.00002	±0.00010	±0.00010	±0.00678	±0	±0.00685
	±0.0071%	±0.0146%	±0.0073%	±0.0071%	±0.0071%	±0.0374%	–	±0.0319%
Server	0	0	0	0	0	0	0	0
Zentrale	±0.000	±0	±0	±0	±0	±0	±0	±0
	–	–	–	–	–	–	–	–
Server	65.29	0.0081	0.0081	0.0485	0.0485	0.6035	0	0.7167
Filiale	±0.034	±0.00001	±0.00000	±0.00002	±0.00002	±0.00046	±0	±0.00051
1	±0.0513%	±0.0903%	±0.0516%	±0.0509%	±0.0509%	±0.0759%	–	±0.0708%
Server	65.30	0.0081	0.0081	0.0485	0.0485	0.6038	0	0.7170
Filiale	±0.034	±0.00001	±0.00000	±0.00002	±0.00002	±0.00046	±0	±0.00051
2	±0.0514%	±0.0902%	±0.0518%	±0.0511%	±0.0511%	±0.0763%	–	±0.0712%
Server	65.30	0.0081	0.0081	0.0485	0.0485	0.6037	0	0.7170
Filiale	±0.034	±0.00001	±0.00000	±0.00002	±0.00002	±0.00046	±0	±0.00051
3	±0.0513%	±0.0911%	±0.0515%	±0.0509%	±0.0509%	±0.0763%	–	±0.0711%
Server	65.28	0.0081	0.0081	0.0485	0.0485	0.6035	0	0.7167
Filiale	±0.033	±0.00001	±0.00000	±0.00002	±0.00002	±0.00046	±0	±0.00051
4	±0.0513%	±0.0924%	±0.0515%	±0.0509%	±0.0509%	±0.0762%	–	±0.0711%
Server	65.31	0.0081	0.0081	0.0485	0.0485	0.6039	0	0.7172
Filiale	±0.033	±0.00001	±0.00000	±0.00002	±0.00002	±0.00046	±0	±0.00051
5	±0.0511%	±0.0908%	±0.0512%	±0.0507%	±0.0507%	±0.0760%	–	±0.0708%
Server	65.31	0.0081	0.0081	0.0485	0.0485	0.6038	0	0.7171
Filiale	±0.033	±0.00001	±0.00000	±0.00002	±0.00002	±0.00046	±0	±0.00051
6	±0.0511%	±0.0919%	±0.0512%	±0.0506%	±0.0506%	±0.0760%	–	±0.0709%

Tabelle C.3: Belastung der WF-Server im verteilten Fall.

	Komm.	WF-Start	WF-Ende	Akt-Start	Akt-Ende	AL-Upd.	Migrat.	Aktionen
Server	65.32	0.0081	0.0081	0.0485	0.0485	0.6041	0	0.7173
Filiale	±0.033	±0.00001	±0.00000	±0.00002	±0.00002	±0.00046	±0	±0.00051
7	±0.0506%	±0.0902%	±0.0511%	±0.0502%	±0.0503%	±0.0757%	–	±0.0705%
Server	65.24	0.0081	0.0081	0.0485	0.0485	0.6031	0	0.7162
Filiale	±0.034	±0.00001	±0.00000	±0.00002	±0.00002	±0.00046	±0	±0.00051
8	±0.0520%	±0.0912%	±0.0523%	±0.0516%	±0.0516%	±0.0760%	–	±0.0710%
Server	65.30	0.0081	0.0081	0.0485	0.0485	0.6037	0	0.7170
Filiale	±0.033	±0.00001	±0.00000	±0.00002	±0.00002	±0.00046	±0	±0.00051
9	±0.0512%	±0.0896%	±0.0516%	±0.0508%	±0.0508%	±0.0762%	–	±0.0710%
Server	65.32	0.0081	0.0081	0.0485	0.0485	0.6040	0	0.7172
Filiale	±0.033	±0.00001	±0.00000	±0.00002	±0.00002	±0.00046	±0	±0.00050
10	±0.0504%	±0.0907%	±0.0508%	±0.0501%	±0.0501%	±0.0754%	–	±0.0703%
Server	65.31	0.0081	0.0081	0.0485	0.0485	0.6039	0	0.7171
Filiale	±0.033	±0.00001	±0.00000	±0.00002	±0.00002	±0.00046	±0	±0.00051
11	±0.0507%	±0.0904%	±0.0511%	±0.0503%	±0.0504%	±0.0763%	–	±0.0711%
Server	65.31	0.0081	0.0081	0.0485	0.0485	0.6039	0	0.7171
Filiale	±0.033	±0.00001	±0.00000	±0.00002	±0.00002	±0.00046	±0	±0.00051
12	±0.0507%	±0.0905%	±0.0509%	±0.0503%	±0.0503%	±0.0757%	–	±0.0706%
Server	65.29	0.0081	0.0081	0.0485	0.0485	0.6035	0	0.7167
Filiale	±0.033	±0.00001	±0.00000	±0.00002	±0.00002	±0.00046	±0	±0.00051
13	±0.0511%	±0.0898%	±0.0513%	±0.0507%	±0.0507%	±0.0761%	–	±0.0709%
Server	65.29	0.0081	0.0081	0.0485	0.0485	0.6037	0	0.7170
Filiale	±0.033	±0.00001	±0.00000	±0.00002	±0.00002	±0.00046	±0	±0.00051
14	±0.0512%	±0.0904%	±0.0516%	±0.0508%	±0.0508%	±0.0762%	–	±0.0711%
Server	65.26	0.0081	0.0081	0.0485	0.0485	0.6032	0	0.7164
Filiale	±0.034	±0.00001	±0.00000	±0.00002	±0.00002	±0.00046	±0	±0.00051
15	±0.0519%	±0.0912%	±0.0522%	±0.0515%	±0.0515%	±0.0767%	–	±0.0716%
Server	65.27	0.0081	0.0081	0.0485	0.0485	0.6035	0	0.7167
Filiale	±0.033	±0.00001	±0.00000	±0.00002	±0.00002	±0.00046	±0	±0.00051
16	±0.0512%	±0.0907%	±0.0515%	±0.0508%	±0.0508%	±0.0757%	–	±0.0706%
Server	65.32	0.0081	0.0081	0.0485	0.0485	0.6039	0	0.7172
Filiale	±0.033	±0.00001	±0.00000	±0.00002	±0.00002	±0.00046	±0	±0.00051
17	±0.0508%	±0.0912%	±0.0511%	±0.0504%	±0.0504%	±0.0757%	–	±0.0705%
Server	65.28	0.0081	0.0081	0.0485	0.0485	0.6036	0	0.7168
Filiale	±0.033	±0.00001	±0.00000	±0.00002	±0.00002	±0.00046	±0	±0.00051
18	±0.0513%	±0.0899%	±0.0516%	±0.0509%	±0.0509%	±0.0760%	–	±0.0708%
Server	65.28	0.0081	0.0081	0.0485	0.0485	0.6036	0	0.7168
Filiale	±0.034	±0.00001	±0.00000	±0.00002	±0.00002	±0.00046	±0	±0.00051
19	±0.0516%	±0.0920%	±0.0518%	±0.0512%	±0.0512%	±0.0766%	–	±0.0714%
Server	65.30	0.0081	0.0081	0.0485	0.0485	0.6039	0	0.7171
Filiale	±0.033	±0.00001	±0.00000	±0.00002	±0.00002	±0.00045	±0	±0.00050
20	±0.0505%	±0.0906%	±0.0507%	±0.0501%	±0.0501%	±0.0752%	–	±0.0701%
Server	65.29	0.0081	0.0081	0.0485	0.0485	0.6036	0	0.7169
Filiale	±0.033	±0.00001	±0.00000	±0.00002	±0.00002	±0.00046	±0	±0.00050
21	±0.0512%	±0.0900%	±0.0513%	±0.0507%	±0.0508%	±0.0754%	–	±0.0704%
Server	65.27	0.0081	0.0081	0.0485	0.0485	0.6035	0	0.7167
Filiale	±0.034	±0.00001	±0.00000	±0.00002	±0.00002	±0.00047	±0	±0.00052
22	±0.0519%	±0.0910%	±0.0522%	±0.0515%	±0.0515%	±0.0771%	–	±0.0719%

Tabelle C.3 (Fortsetzung)

	Komm.	WF-Start	WF-Ende	Akt-Start	Akt-Ende	AL-Upd.	Migrat.	Aktionen
Server	65.30	0.0081	0.0081	0.0485	0.0485	0.6037	0	0.7170
Filiale	±0.033	±0.00001	±0.00000	±0.00002	±0.00002	±0.00045	±0	±0.00050
23	±0.0511%	±0.0909%	±0.0511%	±0.0506%	±0.0506%	±0.0754%	–	±0.0703%
Server	65.31	0.0081	0.0081	0.0485	0.0485	0.6038	0	0.7170
Filiale	±0.033	±0.00001	±0.00000	±0.00002	±0.00002	±0.00045	±0	±0.00050
24	±0.0505%	±0.0903%	±0.0509%	±0.0501%	±0.0501%	±0.0750%	–	±0.0699%
Server	65.32	0.0081	0.0081	0.0485	0.0485	0.6040	0	0.7173
Filiale	±0.033	±0.00001	±0.00000	±0.00002	±0.00002	±0.00045	±0	±0.00050
25	±0.0503%	±0.0901%	±0.0505%	±0.0498%	±0.0498%	±0.0753%	–	±0.0701%
Server	65.30	0.0081	0.0081	0.0485	0.0485	0.6038	0	0.7171
Filiale	±0.033	±0.00001	±0.00000	±0.00002	±0.00002	±0.00046	±0	±0.00051
26	±0.0513%	±0.0918%	±0.0515%	±0.0509%	±0.0509%	±0.0758%	–	±0.0707%
Server	65.32	0.0081	0.0081	0.0485	0.0485	0.6040	0	0.7173
Filiale	±0.033	±0.00001	±0.00000	±0.00002	±0.00002	±0.00046	±0	±0.00051
27	±0.0503%	±0.0910%	±0.0506%	±0.0499%	±0.0499%	±0.0757%	–	±0.0705%
Server	65.34	0.0081	0.0081	0.0485	0.0485	0.6042	0	0.7175
Filiale	±0.033	±0.00001	±0.00000	±0.00002	±0.00002	±0.00046	±0	±0.00051
28	±0.0506%	±0.0906%	±0.0509%	±0.0502%	±0.0502%	±0.0757%	–	±0.0705%
Server	65.26	0.0081	0.0081	0.0485	0.0485	0.6033	0	0.7164
Filiale	±0.034	±0.00001	±0.00000	±0.00003	±0.00003	±0.00047	±0	±0.00052
29	±0.0521%	±0.0907%	±0.0522%	±0.0517%	±0.0516%	±0.0772%	–	±0.0720%
Server	65.33	0.0081	0.0081	0.0485	0.0485	0.6041	0	0.7174
Filiale	±0.033	±0.00001	±0.00000	±0.00002	±0.00002	±0.00045	±0	±0.00050
30	±0.0499%	±0.0904%	±0.0504%	±0.0495%	±0.0495%	±0.0742%	–	±0.0691%

Tabelle C.3 (Fortsetzung)

	Komm.	WF-Start	WF-Ende	Akt-Start	Akt-Ende	AL-Upd.	Migrat.	Aktionen
Summe	1968.98	0.2437	0.2425	1.9402	1.9402	26.9672	0	31.3338
Teil-	±0.140	±0.00004	±0.00002	±0.00014	±0.00014	±0.01327	±0	±0.01334
netze	±0.0071%	±0.0146%	±0.0073%	±0.0071%	±0.0071%	±0.0492%	–	±0.0426%
Summe	10.07	0	0	0.4850	0.4850	8.8550	0	9.8251
Gate-	±0.001	±0	±0	±0.00003	±0.00003	±0.00654	±0	±0.00655
ways	±0.0079%	–	–	±0.0071%	±0.0071%	±0.0738%	–	±0.0666%
Teilnetz	10.07	0	0	0.4850	0.4850	8.8550	0	9.8251
Zentrale	±0.001	±0	±0	±0.00003	±0.00003	±0.00654	±0	±0.00655
	±0.0079%	–	–	±0.0071%	±0.0071%	±0.0738%	–	±0.0666%
Teilnetz	65.29	0.0081	0.0081	0.0485	0.0485	0.6035	0	0.7167
Filiale	±0.034	±0.00001	±0.00000	±0.00002	±0.00002	±0.00046	±0	±0.00051
1	±0.0513%	±0.0903%	±0.0516%	±0.0509%	±0.0509%	±0.0759%	–	±0.0708%
Teilnetz	65.30	0.0081	0.0081	0.0485	0.0485	0.6038	0	0.7170
Filiale	±0.034	±0.00001	±0.00000	±0.00002	±0.00002	±0.00046	±0	±0.00051
2	±0.0514%	±0.0902%	±0.0518%	±0.0511%	±0.0511%	±0.0763%	–	±0.0712%
Teilnetz	65.30	0.0081	0.0081	0.0485	0.0485	0.6037	0	0.7170
Filiale	±0.034	±0.00001	±0.00000	±0.00002	±0.00002	±0.00046	±0	±0.00051
3	±0.0513%	±0.0911%	±0.0515%	±0.0509%	±0.0509%	±0.0763%	–	±0.0711%

Tabelle C.4: Belastung des Kommunikationssystems im verteilten Fall.

	Komm.	WF-Start	WF-Ende	Akt-Start	Akt-Ende	AL-Upd.	Migrat.	Aktionen
Teilnetz	65.28	0.0081	0.0081	0.0485	0.0485	0.6035	0	0.7167
Filiale	±0.033	±0.00001	±0.00000	±0.00002	±0.00002	±0.00046	±0	±0.00051
4	±0.0513%	±0.0924%	±0.0515%	±0.0509%	±0.0509%	±0.0762%	–	±0.0711%
Teilnetz	65.31	0.0081	0.0081	0.0485	0.0485	0.6039	0	0.7172
Filiale	±0.033	±0.00001	±0.00000	±0.00002	±0.00002	±0.00046	±0	±0.00051
5	±0.0511%	±0.0908%	±0.0512%	±0.0507%	±0.0507%	±0.0760%	–	±0.0708%
Teilnetz	65.31	0.0081	0.0081	0.0485	0.0485	0.6038	0	0.7171
Filiale	±0.033	±0.00001	±0.00000	±0.00002	±0.00002	±0.00046	±0	±0.00051
6	±0.0511%	±0.0919%	±0.0512%	±0.0506%	±0.0506%	±0.0760%	–	±0.0709%
Teilnetz	65.32	0.0081	0.0081	0.0485	0.0485	0.6041	0	0.7173
Filiale	±0.033	±0.00001	±0.00000	±0.00002	±0.00002	±0.00046	±0	±0.00051
7	±0.0506%	±0.0902%	±0.0511%	±0.0502%	±0.0503%	±0.0757%	–	±0.0705%
Teilnetz	65.24	0.0081	0.0081	0.0485	0.0485	0.6031	0	0.7162
Filiale	±0.034	±0.00001	±0.00000	±0.00002	±0.00002	±0.00046	±0	±0.00051
8	±0.0520%	±0.0912%	±0.0523%	±0.0516%	±0.0516%	±0.0760%	–	±0.0710%
Teilnetz	65.30	0.0081	0.0081	0.0485	0.0485	0.6037	0	0.7170
Filiale	±0.033	±0.00001	±0.00000	±0.00002	±0.00002	±0.00046	±0	±0.00051
9	±0.0512%	±0.0896%	±0.0516%	±0.0508%	±0.0508%	±0.0762%	–	±0.0710%
Teilnetz	65.32	0.0081	0.0081	0.0485	0.0485	0.6040	0	0.7172
Filiale	±0.033	±0.00001	±0.00000	±0.00002	±0.00002	±0.00046	±0	±0.00050
10	±0.0504%	±0.0907%	±0.0508%	±0.0501%	±0.0501%	±0.0754%	–	±0.0703%
Teilnetz	65.31	0.0081	0.0081	0.0485	0.0485	0.6039	0	0.7171
Filiale	±0.033	±0.00001	±0.00000	±0.00002	±0.00002	±0.00046	±0	±0.00051
11	±0.0507%	±0.0904%	±0.0511%	±0.0503%	±0.0504%	±0.0763%	–	±0.0711%
Teilnetz	65.31	0.0081	0.0081	0.0485	0.0485	0.6039	0	0.7171
Filiale	±0.033	±0.00001	±0.00000	±0.00002	±0.00002	±0.00046	±0	±0.00051
12	±0.0507%	±0.0905%	±0.0509%	±0.0503%	±0.0503%	±0.0757%	–	±0.0706%
Teilnetz	65.29	0.0081	0.0081	0.0485	0.0485	0.6035	0	0.7167
Filiale	±0.033	±0.00001	±0.00000	±0.00002	±0.00002	±0.00046	±0	±0.00051
13	±0.0511%	±0.0898%	±0.0513%	±0.0507%	±0.0507%	±0.0761%	–	±0.0709%
Teilnetz	65.29	0.0081	0.0081	0.0485	0.0485	0.6037	0	0.7170
Filiale	±0.033	±0.00001	±0.00000	±0.00002	±0.00002	±0.00046	±0	±0.00051
14	±0.0512%	±0.0904%	±0.0516%	±0.0508%	±0.0508%	±0.0762%	–	±0.0711%
Teilnetz	65.26	0.0081	0.0081	0.0485	0.0485	0.6032	0	0.7164
Filiale	±0.034	±0.00001	±0.00000	±0.00002	±0.00002	±0.00046	±0	±0.00051
15	±0.0519%	±0.0912%	±0.0522%	±0.0515%	±0.0515%	±0.0767%	–	±0.0716%
Teilnetz	65.27	0.0081	0.0081	0.0485	0.0485	0.6035	0	0.7167
Filiale	±0.033	±0.00001	±0.00000	±0.00002	±0.00002	±0.00046	±0	±0.00051
16	±0.0512%	±0.0907%	±0.0515%	±0.0508%	±0.0508%	±0.0757%	–	±0.0706%
Teilnetz	65.32	0.0081	0.0081	0.0485	0.0485	0.6039	0	0.7172
Filiale	±0.033	±0.00001	±0.00000	±0.00002	±0.00002	±0.00046	±0	±0.00051
17	±0.0508%	±0.0912%	±0.0511%	±0.0504%	±0.0504%	±0.0757%	–	±0.0705%
Teilnetz	65.28	0.0081	0.0081	0.0485	0.0485	0.6036	0	0.7168
Filiale	±0.033	±0.00001	±0.00000	±0.00002	±0.00002	±0.00046	±0	±0.00051
18	±0.0513%	±0.0899%	±0.0516%	±0.0509%	±0.0509%	±0.0760%	–	±0.0708%
Teilnetz	65.28	0.0081	0.0081	0.0485	0.0485	0.6036	0	0.7168
Filiale	±0.034	±0.00001	±0.00000	±0.00002	±0.00002	±0.00046	±0	±0.00051
19	±0.0516%	±0.0920%	±0.0518%	±0.0512%	±0.0512%	±0.0766%	–	±0.0714%

Tabelle C.4 (Fortsetzung)

	Komm.	WF-Start	WF-Ende	Akt-Start	Akt-Ende	AL-Upd.	Migrat.	Aktionen
Teilnetz	65.30	0.0081	0.0081	0.0485	0.0485	0.6039	0	0.7171
Filiale	±0.033	±0.00001	±0.00000	±0.00002	±0.00002	±0.00045	±0	±0.00050
20	±0.0505%	±0.0906%	±0.0507%	±0.0501%	±0.0501%	±0.0752%	–	±0.0701%
Teilnetz	65.29	0.0081	0.0081	0.0485	0.0485	0.6036	0	0.7169
Filiale	±0.033	±0.00001	±0.00000	±0.00002	±0.00002	±0.00046	±0	±0.00050
21	±0.0512%	±0.0900%	±0.0513%	±0.0507%	±0.0508%	±0.0754%	–	±0.0704%
Teilnetz	65.27	0.0081	0.0081	0.0485	0.0485	0.6035	0	0.7167
Filiale	±0.034	±0.00001	±0.00000	±0.00002	±0.00002	±0.00047	±0	±0.00052
22	±0.0519%	±0.0910%	±0.0522%	±0.0515%	±0.0515%	±0.0771%	–	±0.0719%
Teilnetz	65.30	0.0081	0.0081	0.0485	0.0485	0.6037	0	0.7170
Filiale	±0.033	±0.00001	±0.00000	±0.00002	±0.00002	±0.00045	±0	±0.00050
23	±0.0511%	±0.0909%	±0.0511%	±0.0506%	±0.0506%	±0.0754%	–	±0.0703%
Teilnetz	65.31	0.0081	0.0081	0.0485	0.0485	0.6038	0	0.7170
Filiale	±0.033	±0.00001	±0.00000	±0.00002	±0.00002	±0.00045	±0	±0.00050
24	±0.0505%	±0.0903%	±0.0509%	±0.0501%	±0.0501%	±0.0750%	–	±0.0699%
Teilnetz	65.32	0.0081	0.0081	0.0485	0.0485	0.6040	0	0.7173
Filiale	±0.033	±0.00001	±0.00000	±0.00002	±0.00002	±0.00045	±0	±0.00050
25	±0.0503%	±0.0901%	±0.0505%	±0.0498%	±0.0498%	±0.0753%	–	±0.0701%
Teilnetz	65.30	0.0081	0.0081	0.0485	0.0485	0.6038	0	0.7171
Filiale	±0.033	±0.00001	±0.00000	±0.00002	±0.00002	±0.00046	±0	±0.00051
26	±0.0513%	±0.0918%	±0.0515%	±0.0509%	±0.0509%	±0.0758%	–	±0.0707%
Teilnetz	65.32	0.0081	0.0081	0.0485	0.0485	0.6040	0	0.7173
Filiale	±0.033	±0.00001	±0.00000	±0.00002	±0.00002	±0.00046	±0	±0.00051
27	±0.0503%	±0.0910%	±0.0506%	±0.0499%	±0.0499%	±0.0757%	–	±0.0705%
Teilnetz	65.34	0.0081	0.0081	0.0485	0.0485	0.6042	0	0.7175
Filiale	±0.033	±0.00001	±0.00000	±0.00002	±0.00002	±0.00046	±0	±0.00051
28	±0.0506%	±0.0906%	±0.0509%	±0.0502%	±0.0502%	±0.0757%	–	±0.0705%
Teilnetz	65.26	0.0081	0.0081	0.0485	0.0485	0.6033	0	0.7164
Filiale	±0.034	±0.00001	±0.00000	±0.00003	±0.00003	±0.00047	±0	±0.00052
29	±0.0521%	±0.0907%	±0.0522%	±0.0517%	±0.0516%	±0.0772%	–	±0.0720%
Teilnetz	65.33	0.0081	0.0081	0.0485	0.0485	0.6041	0	0.7174
Filiale	±0.033	±0.00001	±0.00000	±0.00002	±0.00002	±0.00045	±0	±0.00050
30	±0.0499%	±0.0904%	±0.0504%	±0.0495%	±0.0495%	±0.0742%	–	±0.0691%

Tabelle C.4 (Fortsetzung)

C.2.3 Vergleich der Verteilungsmodelle

In Abb. C.1 wird das Simulationsergebnis für die beiden untersuchten Verteilungsmodelle vergleichend dargestellt. Die Abbildung entspricht Abb. 9.12, allerdings mit dem Unterschied, dass die Last in absoluten Zahlen angegeben ist. Die Lastangaben erfolgen also in kByte pro sec bzw. in Aktionen pro sec. Wie aus diesen Werten die relativen Lastangeben berechnet werden, wurde schon in Abschnitt 9.3.4 erläutert. Für den zentralen Fall ist die Last für das Teilnetz des WF-Servers wieder in Klammern oberhalb des Balkens für die durchschnittliche Last pro Teilnetz angegeben.

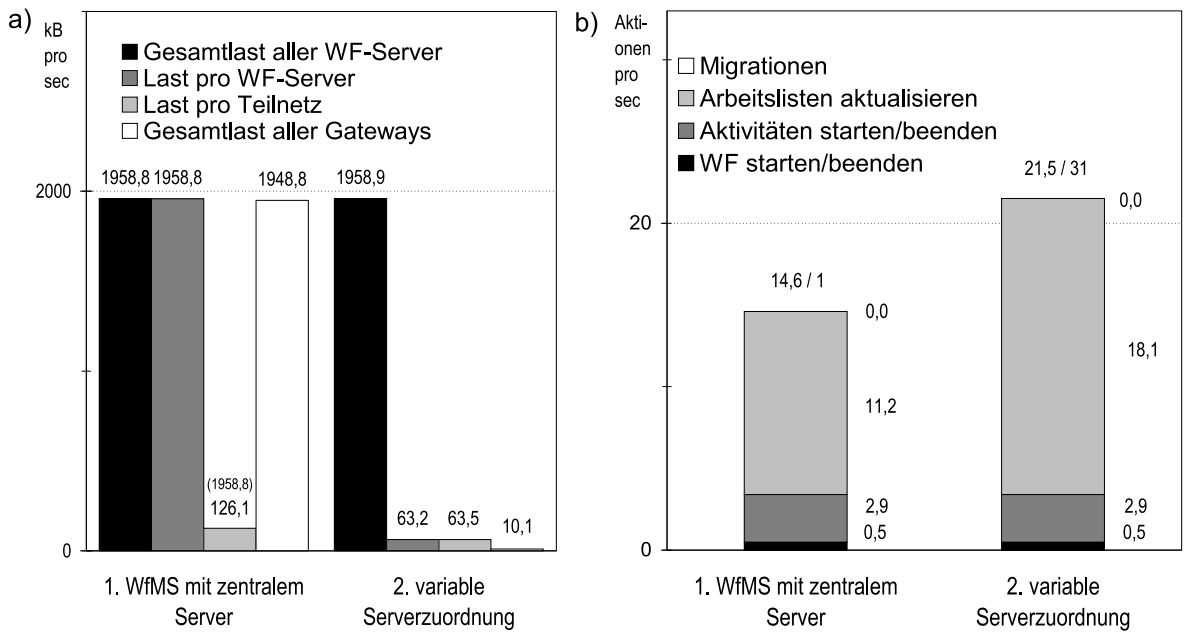


Abbildung C.1 Absolute Lastwerte für das Kreditantragsbeispiel: a) Kommunikation, b) Aktionen.